Evaluating the Impact of Traffic Sampling on AATAC's DDoS Detection

Gilles Roudière and Philippe Owezarski*

LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France
E-mail: gilles.roudiere@gmail.com; owe@laas.fr
**Corresponding Author*

Received 20 November 2018; 20 December 2018;
Publication 20 June 2019

Abstract

As Distributed Denial of Service (DDoS) attack are still a severe threat for the Internet stakeholders, they should be detected with efficient tools meeting industrial requirements. We previously introduced the AATAC detector, which showed its ability to accurately detect DDoS attacks in real time on full traffic, while being able to cope with the several constraints due to an industrial operation, as time to detect, limited resources for running detection algorithms, detection autonomy for not wasting uselessly administrators' time. However, in a realistic scenario, network monitoring is done using sampled traffic. Such sampling may impact the detection accuracy or the pertinence of produced results. Consequently, in this paper, we evaluate AATAC over sampled traffic. We use five different count-based or time-based sampling techniques, and show that AATAC's resources consumption is in general greatly reduced with little to no impact on the detection accuracy. Obtained results are succinctly compared with those from FastNetMon, an open-source threshold-based DDoS detector.

Keywords: DDoS detection, sampled traffic, unsupervised learning.

Journal of Cyber Security and Mobility, Vol. 8_4, 419–438.
doi: 10.13052/jcsm2245-1439.842

1 Introduction

Distributed Denial of Service (DDoS) attacks consist in numerous hosts sending fake requests towards a victim's network. Those requests exhaust the victim's system resources, either its available computing power, networking bandwidth or even some protocol implementation specific resources. Thoses attacks are numerous and they can cause up to the system's total inability to handle legitimate clients' requests. Consequently, they should be detected. However, appropriate detectors have to meet the industry requirements (real-time operation, accuracy, autonomy...)

As today's bandwidths are very large, most actual monitoring systems sample the traffic in order to reduce the size of the data to be processed, limiting the computational resources used for monitoring purposes.[1] Consequently, a realistic DDoS detector should be efficient even when processing sampled traffic.

In a previous paper [1], we introduced AATAC (Autonomous Algorithm for Traffic Anomaly Characterization), a fully autonomous anomaly detector that focuses on DDoS attacks. We proved, on some full traffic traces, that it can perform an efficient detection while consuming a very limited amount of resources. However, to be used in an industrial context, AATAC should also be able to perform efficiently over sampled traces. This is why in this paper we evaluate the AATAC algorithm over sampled traffic. We use several types of sampling techniques and rates, the aim being to understand which sampling techniques would be the most appropriate for maximizing the DDoS detection using AATAC. The results are compared with those from FastNetMon, the only publicly available DDoS detector we manage to install on our testbed.

The remaining of this paper is organized as follows. In Section 2 we introduce other detectors built to operate over sampled traces. Our detector AATAC is briefly introduced in Section 3, while its evaluation over sampled traces is discussed in Sections 4 and 5. Section 6 concludes the paper and discusses possible future works.

2 Related Works

In this section, we present related works that study the impact of sampling on several monitoring tools, but mainly focus on the network anomaly detection.

[1] This processing is often performed on the switches/routers whose main task is to switch/route packets, and are then largely loaded by this main task. Other tasks must then consume resources parsimoniously.

Several related approaches rely on a flow-based sampling. Bartos et al. [2] study the impact of sampling on network anomaly detection. They propose an adaptive flow-level sampling technique that manages to limit the impact of sampling on the performance of a network behavioural anomaly detector. Other flow-based sampling techniques are presented by Jadidi et al. [3] or Andriolakis [4]. The problem with such approaches is that extracting flows from the traffic is already a complex task that has to deal with already packet sampled traffic. The main bottleneck for monitoring system is definitely situated at the packet level, even for systems that provide flow based information. Thus, we instead focus on packet sampling techniques as they are the most likely to significantly impact the computational cost reduction.

A framework to evaluate the impact of packet sampling over several various tools is presented in [5]. The author discusses the very generic performances of each sampling algorithm. They propose a set of metrics that allow the evaluation of each technique's ability to produce a sampled traffic that efficiently represents the original one. Their results are really generic and do not target the network anomaly detection problem specifically, they thus might not apply in our situation. An extension of this study is presented in [6].

Jun et al. [7] propose an adaptive sampling technique that intends to keep the amount of sampled traffic below a maximum inspection capability. They use the SDN technology to distribute the sampling over several switches. They perform their evaluation using Snort [8] and Suricata [9], two detectors based on Deep Packet Inspection. As such techniques focus on the packets content, instead of more encompassing statistical features of the traffic, the technique might not suit all detection algorithms (including AATAC).

In an older paper, Brauckhoff et al. [10] evaluate the impact of sampling over anomaly detection metrics. Their evaluation uses traces containing the Blaster worm to evaluate several detection techniques at various sampling rates. With a detector they propose, and thanks to an entropy-based summarization technique, authors achieve a good detection independently from how high is the sampling rate. The paper also shows that flow-based detection is more impacted by the packet sampling than packet-based detection.

3 AATAC Algorithm Overview

AATAC is a balanced solution to the DDoS detection problem. It provides a real-time detection with low computing resources while still producing pertinent and eloquent results. It is a fully autonomous detector relying on unsupervised machine learning techniques, requiring very few configuration

or updates. In this section we present only a brief overview of the algorithm. For more technical details, please refer to the original publication [1].

AATAC's processing is illustrated on Figure 1. It is split into two distinct parts: a continuous and a discrete one. The first one quickly handles instances and maintain a data structure representing the traffic in real time. The second one uses this data structure to store, at a regular interval, a snapshot of the traffic. Those snapshots are then stored and used to detect anomalies in the traffic. As those snapshots can be plotted into a set of two-dimensional graphs, they provide the network administrator with a dynamic and pertinent view of the traffic properties when the anomaly occurs.

3.1 Continuous Processing

The first step in AATAC's processing consists in extracting per-flow aggregated data from the trafic. The flow aggregation relies on a short thumbling window that groups the packets according to the 5-tuple (IP_{source}, $IP_{destination}$, $Port_{source}$, $Port_{destination}$, $Protocol$). Each flow's 5-tuple is then recorded alongside with a timestamp value (set at the start of the thumbling window) and a set of various features associated with the flow. Such features include, for example, the number of UDP packets in the flow, the average number of bytes per packet or the number of SYN packets. The per-flow aggregation aims at ensuring AATAC's compatibility with common flow export technologies, such as Netflow [11] or IPFix [12].

Once aggregated, those flows are fed into AATAC's continuous processing. This part of the algorithm builds a representation of the traffic, accurate at any time by reliying on a statistical analysis on the traffic. Its processing is inspired from D-Stream [13], a grid-based clustering algorithm.

To characterize the traffic properties, AATAC uses grids. Those grids are either grouped as histograms, to characterize a traffic feature distribution, or alone, to characterize a global feature of the traffic. Whenever AATAC processes a traffic instance (here a traffic flow and its associated characteristics), AATAC determines a set of grids it falls in. The algorithm then calculate a density assigned to the grid calculated from all instances that fell into the grid so far.

To calculate the grid's density value, AATAC assigns a density to each instance. Let us consider an instance x that fells into a grid at a given time t_x. It's density at a later time t is noted $D(x,t)$, and is calculted according to the following formula:

$$D(x,t) = w_x \lambda^{t-t_x} \tag{1}$$

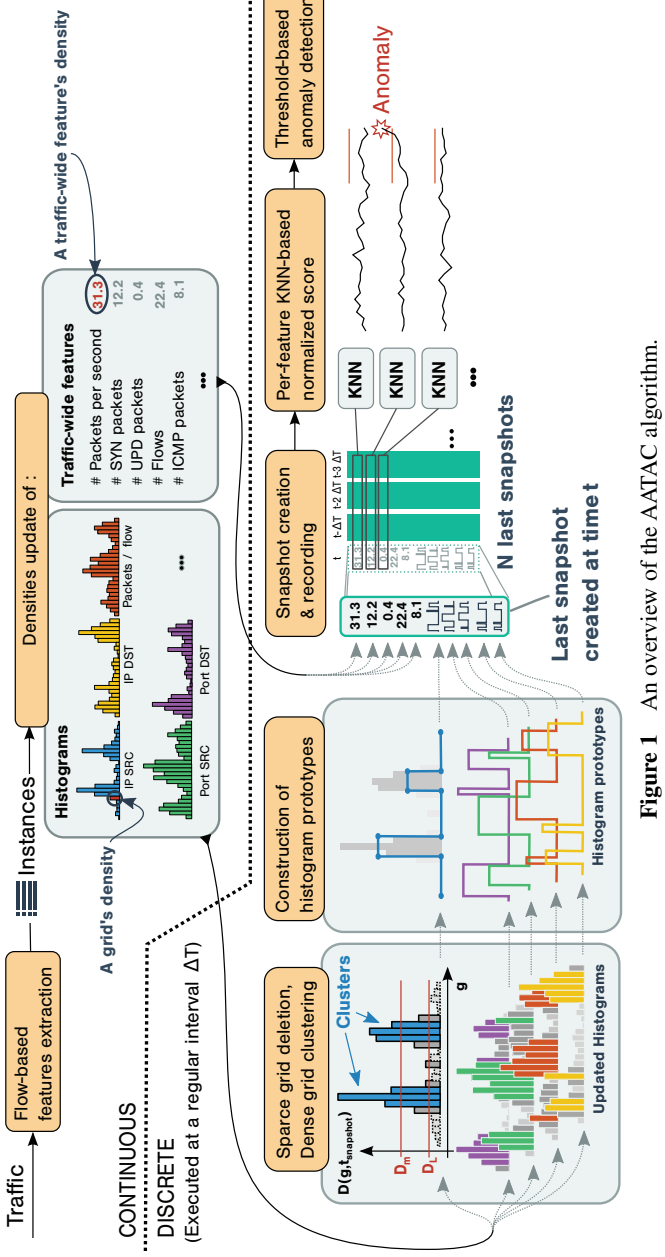


Figure 1 An overview of the AATAC algorithm.

Where w_x is a weight assigned to the instance, that depends on what the grid is characterizing. For example, a per-packet characterization of the traffic requires w_x to be set to the number of packets in the flow. Over time, the density of each instance exponentially decays towards 0. The λ parameter, ranging between 0 and 1, determines how fast those densities decrease. This parameter is called the *decay factor*.

Considering the set $E(g,t)$ of all instances that fell into the grid g at a time t. The density assigned to g is then calculated as follows :

$$D(g,t) = \sum\nolimits_{x \in E(g,t)} D(x,t) \tag{2}$$

The most interesting part of this instance weighting strategy is that the grid's density can be updated in an incremental manner. This limits the memory used, as it is not required to store a large amounts of instances to compute the grid's density. Indeed, to incrementally compute a grid's density, only two values need to be stored: the timestamp of the last instance included into the grid (t_l) and the grid's density at this time ($D(g,t_l)$). When a new instance is added at a time t_n, the grid's density is updated as follows:

$$D(g,t_n) = \lambda^{t_n - t_l} D(g,t_l) + w_n \tag{3}$$

where w_n is the weight assigned to the new instance.

This incremental weighting technique inherently gives much more weight to recent instances in the model than to the older ones. Thus, at a given instant, all grids together form a short-term complete characterization of the traffic. Also, unlike techniques relying on a thumbling window, this approach does not require to wait for the end of a window to extract the traffic properties. This allows a faster detection, which is essential to DDoS attacks detection.

For reasons made explicit in [1], the λ decay factor is set via the R parameter. We define R as $R = \lambda^{\Delta T}$, where ΔT is the parameter defining the time interval between two snapshot creations.

3.2 Discrete Processing

The discrete processing is executed at a regular interval ΔT, it can be split into three steps. First, it updates the continuous processing data structure, it then creates a traffic snapshot and finally detects anomalous behaviours.

As the continuous processing is incremental, the whole set of grids' densities might not create a consistent characterization at a given instant. The grids'

densities are thus updated to be consistent with each other as follows:

$$D(g,t) = \lambda^{(t-t_l)} D(g,t_l) \tag{4}$$

Also, to avoid the number of stored grids to overgrow in the memory, several grids having a density lower than a really low value D_l are removed from the data structure. Indeed, grids that have a very low density did not received any instance for a very long time, and are thus not useful anymore to create a short-term characterization of the traffic.

For performance reasons, AATAC's data structure cannot be stored as is. Grids organized as histograms are thus simplified, and transformed into histogram prototypes, a lighter and faster to process data structure. Those prototypes are created by selecting the most dense grids in each histogram (with a density above a threshold D_m). Adjacents dense grids are then grouped as clusters, whose properties (average density, minimum and maximum boundaries) are then used to create the piecewise constant curve constituting the histogram prototype.

The final traffic snapshot is created by storing all histogram prototypes along with the up-to-date densities characterizing global traffic features. This set constitutes the *snapshot features*.

Finally in the anomaly detection phase, AATAC compares the lastly created snapshot to the set of N last snapshots. This is done using the k-Nearest Neighbour (kNN) algorithm applied for each analyzed snapshot feature. To compare two histogram prototypes, a distance function computes the area that is not shared between the two areas under each prototype's curve (Figure 2 illustrates this calculation). The absolute difference is used for global densities values. Finally, kNN produces an outlierness score that AATAC uses to determine how different is the lastly created snapshot from the previously created ones. Note that the scores are normalized, so that the per-feature produced score can be compared to each other.

The final anomaly detection is performed by detecting if the kNN score goes over a given threshold for any snapshot feature. If so, an alarm is raised. As the histogram prototypes and the global densities can be plotted, this alarm is given to the network administrator along with a set of graphs allowing him to have a pertinent and dynamic view of the traffic features when the anomaly occurs. Figure 3 shows an example of such visual feedback. On this example, the figure depicts each snapshot feature's value (as an histogram or a numerical value) along with the current anomaly score associated with this feature. Scores that went above the threshold have their background colored in red.

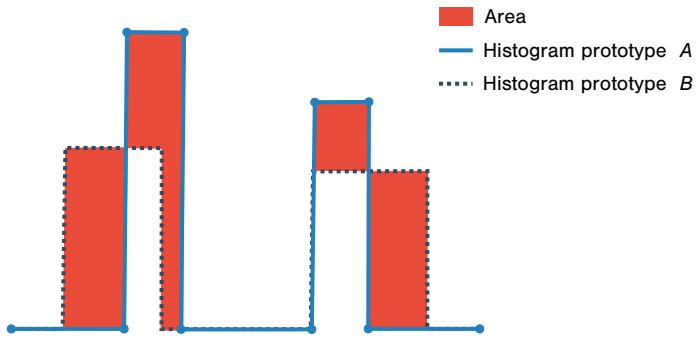


Figure 2 Area computed by the distance function to compare two histogram prototypes.

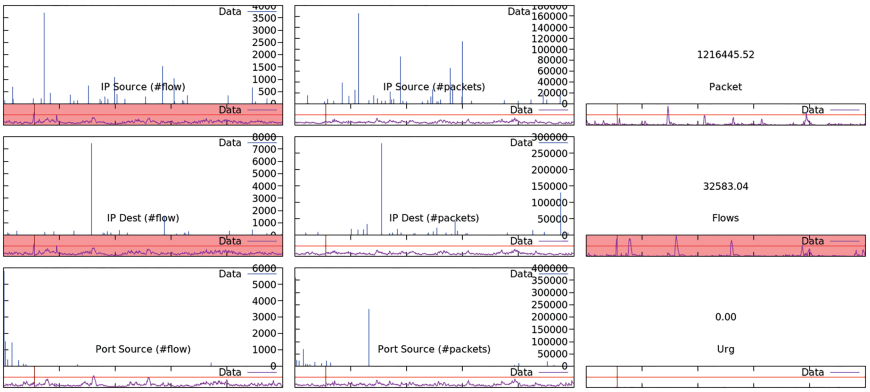


Figure 3 A set of graphs that can be shown to the administrator.

4 Evaluating the Impact of Sampling on the DDoS Detection

In this section, we describe the testbed we used to evaluate AATAC's performances over sampled traffic.

4.1 Sampling Techniques

Basically, a packet sampling algorithm aims at selecting whether or not a packet should be sampled for further analysis. To evaluate AATAC's performances in various situations, we used five different sampling techniques with several sampling rates: *systematic count-based*, *probabilistic*, *1-out-of-N*, *systematic time-based* and *random time-based*. Those techniques can be split into two categories: count-based or time-based. They cover the full range of common techniques for traffic sampling.

4.1.1 Count-based selection techniques

Count-based selection techniques give all packets the same probability to get sampled, independently from their content or arrival time. Statistically, this implies selecting one packet out of N.

The first approach we use is the systematic approach. It consists in systematically selecting a packet every N packets. This approach is pretty straightforward to implement and gives an accurate representation of the traffic in most situations. However, it might suffer from a bias if the monitored features exhibit a periodic behaviour.

The random probabilistic sampling runs a random test whenever a packet arrives and selects it with a given probability p. As it is does not require an analysis of the packet's content, our approach uses a uniform probabilistic sampling that selects each packet with the same probability.

The last random count-based technique we use is the *n-out-of-N* sampling. Every N packets, this technique chooses a random set of n indexes between 0 and $N-1$. Those indexes correspond to the indexes of the n packets that will be selected among the next N packets. In most cases, n is set to 1. The *1-out-of-N* sampling is easier to implement while showing similar performances as any other n values.

Count-based techniques have the advantage to produce a sampled output that follows the trend of the input in terms of number of packet. They thus allow detecting significant changes that depend directly on the number of packets per second in the traffic.

4.1.2 Time-based techniques

Time-based techniques operate by defining a set of dates when a packet selection should be triggered. They generally do that by computing a waiting time interval between each selection. Unlike count-based sampling, time-based sampling does not produce an output that is proportional to the input in terms of number of packets. They instead smooth the variations of the traffic, which might not suit all detection techniques.

If the time between two selections is constant, such approach is said systematic. Like the count-based systematic approach, this technique might suffer from a bias if the traffic exhibits a periodic behaviour.

Other time-based techniques might randomly select the time to wait between two triggers. The technique we use generates an inter-selection interval following an exponential distribution.

Unlike count-based sampling, time-based sampling does not produce an output that is proportional to the input in terms of number of packets. Those

approaches tend to smooth the variations of the traffic, which might not suit all detection techniques.

4.1.3 Other techniques

Various other techniques have been proposed in the literature. For example, adaptive sampling techniques adapt the sampling rate depending on the load of the system (CPU, memory, bandwidth...). Other techniques, said hybrid, use a combination of two other techniques. While being interesting, those techniques do not suit our requirements as they introduce several other parameters that would have made difficult a generalization of the results. Also, such techniques are in fact not common in an industrial context. They require either a knowledge database, or setting up a feedback loop from a performance monitoring system to the sampling hardware, which is not easy to set up and maintain.

4.2 The Testbed

To evaluate the performances of AATAC, we implemented it using C++. The experiments are run on a standard computer, featuring a 3.00 GHz Intel Xeon CPU (E5-2623 v3). It features 8 cores but our implementation does not handle multi-threading.

To evaluate our algorithm's capability to perform an accurate detection, we needed a ground trust. Out of all publicly available datasets we could find, none of them could perfectly fit the needs of our evaluation. Built from a simulated network in which attacks were manually generated, the KDD99 dataset [14] has been widely used in the litterature. Despite its quality, this dataset is now almost 20 years old, and cannot be considered as realistic enough regarding today's trafic. The MAWILab dataset [15] consists in a set of labels associated to the MAWI traces. Captured at the entrance of cloud service provider, those traces are recent and representative of today's traffic. However, the MAWILab labels are generated using a combination of several detectors, which are, by nature, not reliable. The most promising dataset we could find was the UNB ISCX Intrusion Detection Evaluation DataSet [16]. This dataset from 2012 consists in artificially generated traffic, built from the analysis of real captured traces. Four attack patterns are then played during the traces generation, two DDoS, a scan and a brute force attack. Despite being interesting for our evaluation, we consider that this dataset is insufficently diversified for a pertinent evaluation of the detector. As a consequence of the lack of pertinent available dataset, we created our own.

Thus, in the context of the ONTIC project, we built the SynthONTS labelled dataset to evaluate our algorithm. It contains a one hour long, payload-free and anonymized real traces captured at the entrance of a large cloud service provider. In an emulated network, we generated 12 realistic attacks and inserted them into the dataset. This set mainly includes DDoS attacks, such as fraggle attacks, smurf attacks, Syn flooding and UDP flooding attacks. Table 1 lists those attacks and their main properties. The dataset is publicly available on the website of the ONTIC project [17].

We sampled the traces thanks to a custom tool called PCAPsampler [18]. For count-based algorithms, we used the sampling rate of one packet out of 500, 1000, 2000, 5000 or 10000 packets. For time-based algorithms, we picked one packet every 1, 0.1, 0.01 or 0.001 seconds. As the average packet rate is around 67 k packets per-seconds in the dataset, this corresponds to values between picking one packet out of 67000, to one out of 67. According to our industrial partners, a sampling rate of one out of 2000 is a common value used on operation. Those value ranges were thus chosen considering this information, but still wide enough for significantly covering different sampling rates.

Regarding the parameters of AATAC, we used the parameters exhibiting the better performances in our previous evaluation [1]. Thus, we set $N = 500$, $\Delta T = 1\ s$ and $R = 0.9$. To adapt to the sampling rate, the computed instances weights are multiplied according to the sampling rate.

Table 1 List of attacks inserted into the SynthONTS dataset

Attack Type	Generation Tool	Duration	Number of Packets	Average Bandwidth
DDoS Fraggle	nmap	3:55	28 k	9 kB/s
DDoS Fraggle	nping	3:37	7104 k	259 MB/s
DDoS Smurf	hping3	1:33	12645 k	143 MB/s
DDoS Smurf	nping	3:51	1141 k	5.26 MB/s
DDoS Smurf	nping	2:31	6826 k	47 MB/s
DDoS Syn flooding	hping3	2:07	4796 k	21 MB/s
DDoS Syn flooding	nping	2:53	7247 k	16 MB/s
DoS Syn flooding	hping3	3:38	5228 k	17 MB/s
DoS Syn flooding	nping	2:58	7053 k	10 MB/s
DoS UDP flooding	nping	2:30	4127 k	24 MB/s
FTP brute force cracking (rockyou.txt)	ncrack	4:28	5373 k	1,53 MB/s
FTP brute force cracking (500-worst-password.txt)	ncrack	2:07	2558 k	1,55 MB/s

While our implementation of AATAC is able to directly operate from pcap files, this is not possible with FastNetMon. We thus had to replay the whole set of traces (the 1h long traces multiplied by the 12 attacks to evaluate) to evaluate its accuracy. As this requires some time, we only evaluated FastNetMon's performances with the *1-out-of-N* sampling algorithm considering a sampling rate of one packet out of 2000.

4.3 Evaluation Methodology

To measure the accuracy of the detector, we run both implementations over the sampled traces. We consider an alarm as a true positive if it is raised while the anomaly is occurring, a false positive if it is not. To analyse the results, we use the well known Receiver Operating Characteristic (ROC) curve. However, as they suffer from the *base-rate fallacy* issue [19], we complete our evaluation by plotting the IDS operation curve of the detector. This method proposed by Nasr et al. [20], consists in plotting the *positive predictive value* (PPV) along with the false positive rate. Each plotted curve is compared to the *zero reference curve (*ZRC*)*, the operation curve of an ideal detector that should detect all anomalies while producing an increasing number of false positives.

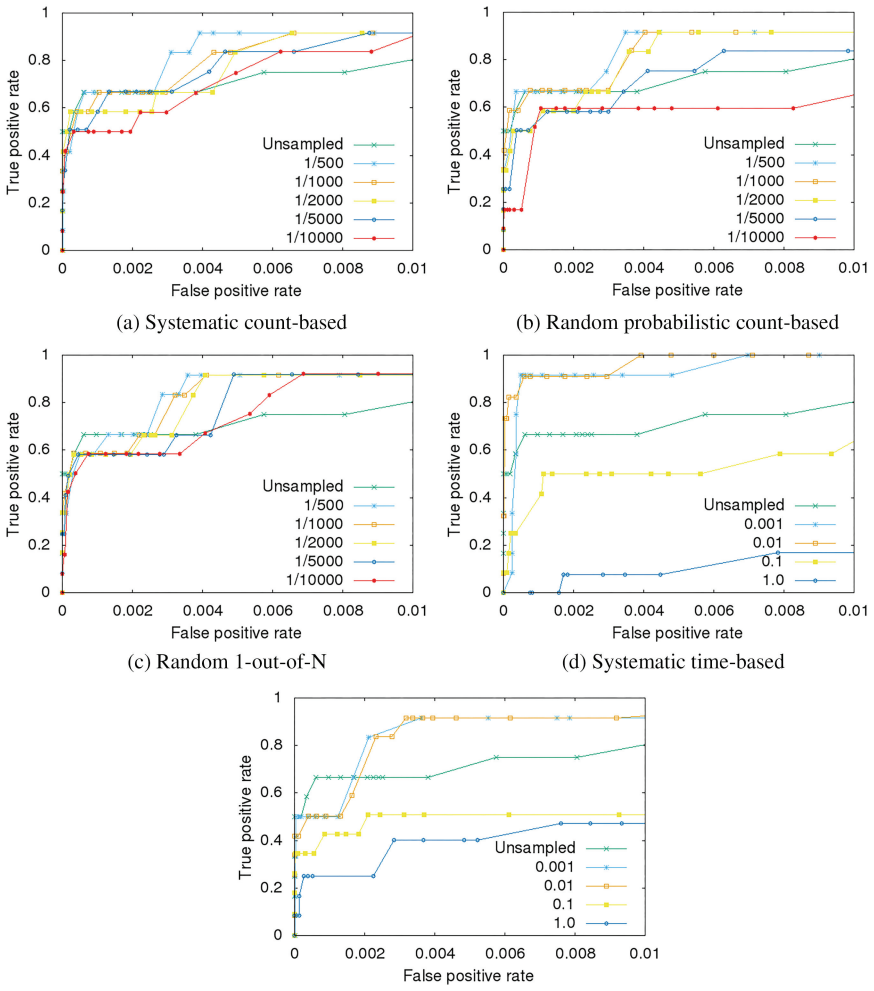
The *intrusion detection effectiveness* ($E_{ID} \in [0, 1]$), is then extracted from this curve. It corresponds to the normalized area between the actual IDS operation curve and the ZRC for a FPR between 0 and a maximum acceptable false positive rate T_{FP}. The lower the E_{ID}, the more effective the detector.

To evaluate how sampling impacts the computational resources required to run the algorithm, we measure how much time our implementation spends into either the continuous or the discrete parts of the algorithm. For the continuous part, we plot the time required to process one second of traffic, and for the discrete part, we measure the time required to create a single snapshot.

5 Results

5.1 Detection Accuracy

Figure 4 shows the ROC curves we obtain for the five sampling algorithms. All count-based sampling techniques achieve good results, but the 1-out-of-N technique seems to produce better and more stable results than other ones. From the time-based sampling techniques, the systematic sampling achieves the best results. For high sampling rates, the systematic time-based approach seems to produce better results than the unsampled traffic while for lower



(a) Systematic count-based

(b) Random probabilistic count-based

(c) Random 1-out-of-N

(d) Systematic time-based

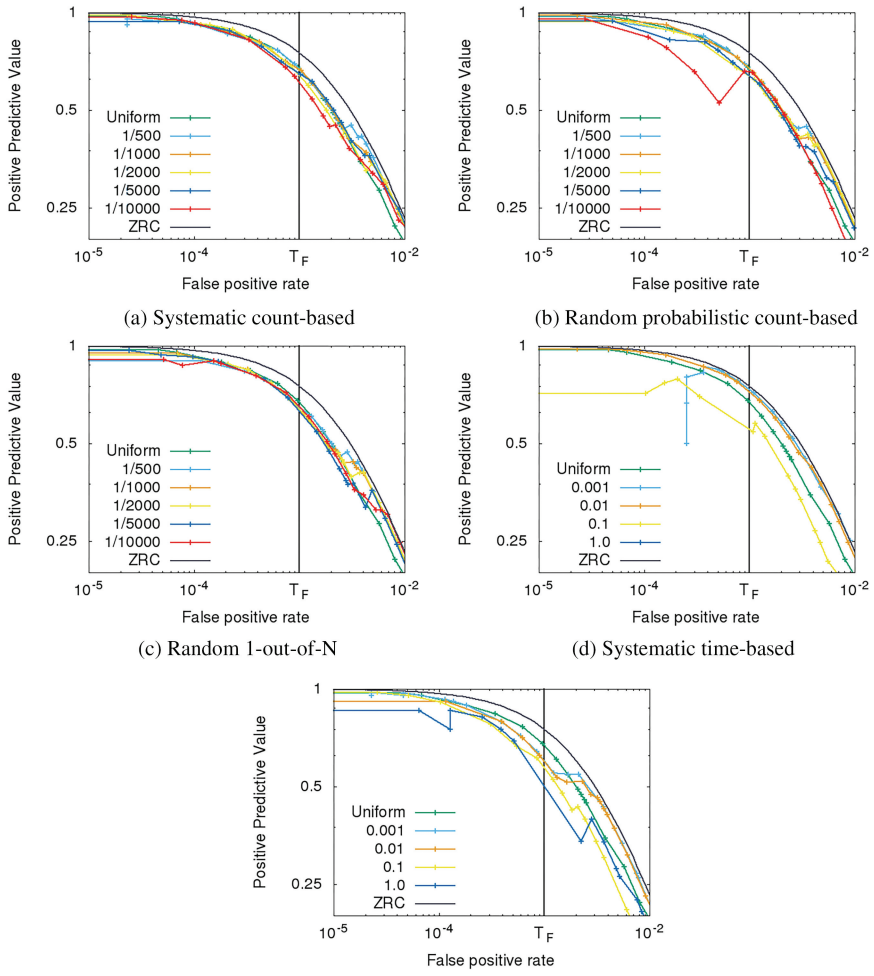
(e) Random time-based

Figure 4 ROC curves for the several sampling algorithms and multiple sampling rates.

sampling rates, with one packet every $100ms$ or $1s$, the results are not as good. Indeed, those sampling rates correspond to one packet every 6700 or 67000 packets. The count-based sampling algorithms produce better results for similar sampling rates, as they detect more attacks for equivalent false-positive rate.

As we can see from the curves, once the false-positive value reaches a given value, AATAC seems to perform better with the sampled traffic than with the unsampled one. As stated before, we multiplied the weight of instances depending on the sampling rate. This has increased the detector's sensitivity.

That being said, the results are more significant with the lowest values of the FPR. Indeed, as we run a test every second, even a FPR of 0.001 means raising a false positive every 17 minutes, which is not acceptable in a real situation. With very low false positive rates, such as 10^{-4}, AATAC performs better with unsampled traffic or high sampling rates. The IDS curves in Figure 5 confirm this analysis. Considering the PPV values for acceptable false-positive rate (below $T_{FP} = 10^{-3}$), AATAC's with unsampled



(a) Systematic count-based

(b) Random probabilistic count-based

(c) Random 1-out-of-N

(d) Systematic time-based

(e) Random time-based

Figure 5 IDS operation curves for the several sampling algorithms and multiple sampling rates.

traffic IDS operation curves stays above the other ones. However, regardless of the sampling algorithm, the efficiency decrease stays very low. This is shown in Table 2: the Intrusion Detection Effectiveness only increases by a very low amount when the sampling rate reduces. This table also confirms that the *1-out-of-N* technique has the most stable results, especially for $T_{FP} = 10^{-3}$.

From those results, time-based sampling stands out. With high sampling rates, AATAC achieves a detection that is exceptionally good, looking better than with unsampled traffic. This may be explained by the fact that time-based sampling tends to smooth out the short term variations of the traffic, reducing the FPR. Also, AATAC relies on densities that depend on the number of packets received per second. Making this rate constant makes the detection less dependent on the packet rate itself, and more on other properties of the traffic. This has probably enhanced the detection.

Regarding the impact of sampling over FastNetMon accuracy, our evaluation showed that FastNetMon was completely unable to perform its detection over sampled traffic. Indeed, with standard sampling rate of one packet out of 2000, FastNetMon generated 7 false positives in our evaluation, while detecting no attacks out of 12. Decreasing the threshold only increased the number of false positives generated by FastNetMon. The detector's inability to detect DDoS is probably due to the fact it operates at an IP address level.

Table 2 Intrusion Detection Effectiveness for all [2] *sampling techniques* and several values of T_{FP}

		Systematic Count-based				
T_{FP}	Unsampled	1/500	1/1000	1/2000	1/5000	1/10000
0.01	0.1543	0.0853	0.1023	0.1197	0.1051	0.1568
0.001	0.0680	0.1026	0.0819	0.0789	0.0958	0.1053
0.0001	0.0155	0.2602	0.0208	0.0163	0.0263	0.0256

	1-out-of-N					Probabilistic				
T_{FP}	1/500	1/1000	1/2000	1/5000	1/10000	1/500	1/1000	1/2000	1/5000	1/10000
0.01	0.0804	0.0903	0.0968	0.1371	0.1204	0.0758	0.0822	0.0909	0.1285	0.2127
0.001	0.0910	0.0804	0.0851	0.0902	0.0899	0.0661	0.0649	0.1014	0.1092	0.2328
0.0001	0.0286	0.0197	0.0274	0.0351	0.0654	0.0232	0.0136	0.0276	0.0390	0.0610

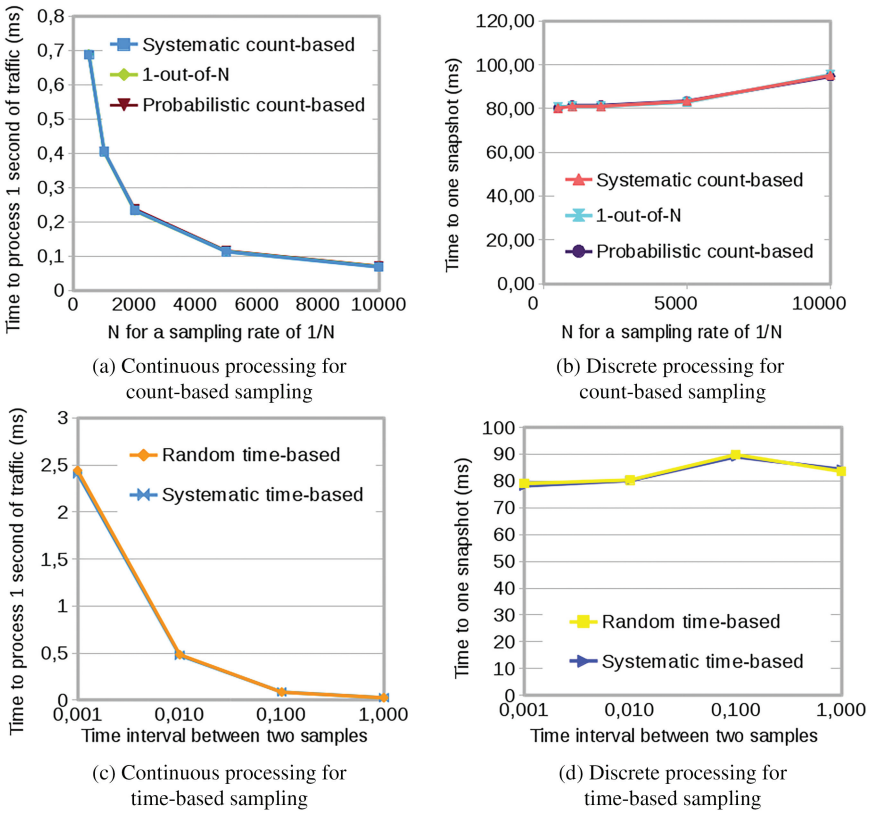
	Systematic Time-based				Random Time-based			
T_{FP}	1ms	10ms	100ms	1s	1ms	10ms	100ms	1s
0.01	0.0631	0.0394	0.2959	0.8424	0.0851	0.0939	0.2954	0.2402
0.001	0.2308	0.0242	0.2113	1.0000	0.1364	0.1127	0.1580	0.1607
0.0001	0.8993	0.0101	0.1238	1.0000	0.2495	0.0207	0.0274	0.0927

Indeed, at such sampling rates, the amount of data corresponding to a specific IP address is too low for the detector to take a pertinent decision. This implies raising more false positive.

As a conclusion, we can see that AATAC performs an accurate detection independently for the sampling technique used. It even performs better at high sampling rates with time-based sampling. The sampling rate has a very limited impact on the quality of AATAC's detection, which is not the case for FastNetMon, which was totally unable to operate with sampled traffic at a standard sampling rate.

5.2 Computing power requirements

The calculated processing time of AATAC for each sampling algorithm are depicted on Figure 6. To process the traffic in real time, the continuous



(a) Continuous processing for count-based sampling

(b) Discrete processing for count-based sampling

(c) Continuous processing for time-based sampling

(d) Discrete processing for time-based sampling

Figure 6 AATAC's processing times for several sampling techniques and sampling rate

processing was designed to process the traffic with a linear complexity regarding the number of instances. This is confirmed on Figures 6(a) and (b). As the X axis corresponds to the inverse of how many packets are processed per second, the processing time follows a $x \mapsto \frac{1}{x}$ pattern.

Regarding the discrete processing, we can see on Figures 6(c) and (d) that the sampling has little impact on the time required to create a snapshot. Indeed there is no specific reason why the treatment should be more complex with a snapshot built from a sampled traffic than with unsampled ones. However, we can see a slight increase of the processing time when few packets per second are sampled. This is probably due to the fact that a stronger sampling implies that the impact of each instance has to be higher on the model. This thus produces a more varying characterization of the traffic which might create more complex histogram prototypes. This impact is however minimal.

We can also see from the several curves that only the sampling rate has an impact on the processing time, while the sampling technique has none.

This evaluation proved that using AATAC with sampled traffic is possible and allows a significant resource consumption reduction, proportional to the sampling rate.

6 Conclusion

In this paper, we evaluated the AATAC DDoS detector's performances over sampled traffic. For sampling rates going up to 1 packet out of 2000, AATAC has shown an almost unchanged detection accuracy while benefiting from a substantial reduction of the required computational resources. From the several sampling techniques used, count-based sampling techniques produce results less dependent on the sampling rate (especially for the *1-out-of-N* technique). However, at very high sampling rates, time-based sampling techniques tend to produce better results, as they smooth the short-term variations of the traffic. For further studies, it thus would be interesting to study a new sampling technique. An ideal one would probably use an adaptive mechanism, combining both time-based sampling for high sampling rates and count-based for lower ones.

References

[1] G. Roudière and P. Owezarski, "A Lightweight Snapshot-Based DDoS Detector," in *2017 13th International Conference on Network and Service Management (CNSM)*, 2017.

[2] K. Bartos, M. Rehak, and V. Krmicek, "Optimizing flow sampling for network anomaly detection," in *IWCMC 2011 - 7th Int. Wirel. Commun. Mob. Comput. Conf.*, pp. 1304–1309, 2011.
[3] Z. Jadidi, V. Muthukkumarasamy, E. Sithirasenan, and K. Singh, "A Probabilistic Sampling Method for Efficient Flow-based Analysis," *J. Commun. Networks*, vol. 18, no. 5, pp. 818–825, 2016.
[4] G. Androulidakis and S. Papavassiliou, "Intelligent flow-based sampling for effective network anomaly detection," in *GLOBECOM – IEEE Glob. Telecommun. Conf.*, pp. 1948–1953, 2007.
[5] J. M. C. Silva, P. Carvalho, and S. R. Lima, "A Modular Sampling Framework for Flexible Traffic Analysis," 2015.
[6] J. M. C. Silva, P. Carvalho, and S. R. Lima, "Analysing traffic flows through sampling: A comparative study," in *Proc. - IEEE Symp. Comput. Commun.*, vol. 2016-Feb., pp. 341–346, 2016.
[7] J.-h. Jun, D. Lee, and S.-h. Kim, "DDoS Attack Detection Using Flow Entropy and Packet Sampling on Huge Networks," *Thirteen. Int. Conf. Networks.*, no. c, pp. 185–190, 2014.
[8] M. Roesch, "Snort: Lightweight Intrusion Detection for Networks.," *LISA '99 13th Syst. Adm. Conf.*, pp. 229–238, 1999.
[9] "suricata." https://suricata-ids.org/
[10] D. Brauckhoff, B. Tellenbach, A. Wagner, M. May, and A. Lakhina, "Impact of packet sampling on anomaly detection metrics," *Proc. 6th ACM SIGCOMM Conf. Internet Meas.*, pp. 159–164, 2006.
[11] B. Claise, "Cisco systems netflow services export version 9," RFC 3954, RFC Editor, October 2004.
[12] J. Quittek, T. Zseby, B. Claise, and S. Zander, "Requirements for ip flow information export (ipfix)," RFC 3917, RFC Editor, October 2004.
[13] Y. Chen and L. Tu, "Density-Based Clustering for Real-Time Stream Data," in *Proc. 13th ACM SIGKDD Int. Conf. Knowl. Discov. data Min.*, pp. 133–142, 2007.
[14] U. K. Archive, "KDD Cup 1999 Data." http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html. Accessed: 2018-01-24.
[15] R. Fontugne, P. Borgnat, P. Abry, and K. Fukuda, "MAWILab : Combining Diverse Anomaly Detectors for Automated Anomaly Labeling and Performance Benchmarking," in *Proc. 6th Int. Conf. Emerg. Netw. Exp. Technol. Co-NEXT'10*, 2010.
[16] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Comput. Secur.*, vol. 31, no. 3, pp. 357–374, 2011.

[17] "Ontic." http://ict-ontic.eu/. Accessed: 2017-05-12.
[18] "Pcapsampler." https://github.com/groud/pcapsampler.
[19] M. Bar-Hillel, "The Base-Rate Fallacy In Probability Judgments," *Acta Psychol. (Amst).*, vol. 44, no. 3, pp. 211–233, 1980.
[20] K. Nasr, A. A.-e. Kalam, and C. Fraboul, "Performance Analysis of Wireless Intrusion Detection Systems," in *Internet Distrib. Comput. Syst. 5th Int. Conf. IDCS 2012, Wuyishan,*, pp. 238–252, 2012.

Biographies

Gilles Roudière received his PhD from Université de Toulouse in 2018. He prepared it at LAAS (Laboratory for Analysis and Architecture of Systems), in Toulouse, France. As his field of research relates to Internet security issues, he is currently working on building a new network anomaly detector that provides a more autonomous detection. His researches lead him to investigate techniques that are able to deal with networks big data, such as machine learning and data mining.

Philippe Owezarski is director of research at CNRS (the French center for scientific research), working at LAAS (Laboratory for Analysis and Architecture of Systems), in Toulouse, France. He got a PhD in computer science in 1996 from Paul Sabatier University, Toulouse III, and an habilitation

for advising research in 2006. His main interests deal with next generation Internet. More specifically Philippe Owezarski takes advantage of IP networks monitoring for enforcing Quality of Service and security. It especially focuses on techniques as machine learning and data mining on the big data collected from the networks for making the network related analytics autonomous and cognitive.