Scan Detection under Sampling: A New Perspective

Ignasi Paredes-Oliva*, Pere Barlet-Ros* and Josep-Solé-Pareta*
*Dept. of Computer Architecture
Universitat Politècnica de Catalunya BarcelonaTech (UPC)
Campus Nord, Edif. D6, C. Jordi Girona, 1-3, 08034 Barcelona, Spain

***Abstract*—Nowadays, due to current high-speed links, applying traffic sampling has become nearly mandatory in order to make Internet traffic monitoring feasible. However, its impact on state-of-the-art algorithms is unclear and has become a topic of foremost importance. Specifically, focusing on the impact of sampling on the detection of scanning cyberattacks, former studies concluded that *Flow Sampling* was the best technique. In this paper we first evaluate how two well-known algorithms for scan detection perform under sampling and confirm its dramatic impact. Unlike previously reported, we show that *Packet Sampling* performs better than *Flow Sampling* under certain scenarios. This is important because routers only support packet-based sampling. The second part of this paper, taking into account the good results for scan detection reported by a sampling technique called *Selective Sampling (SES)*, proposes a new sampling technique, *Online Selective Sampling (OSES)*, that samples the same traffic that *SES* but uses less resources. Instead of requiring aggregation of packets into flows before sampling, *OSES* works online on a packet-per-packet basis and, therefore, it does not need to capture all the traffic. We show that *OSES* is significantly faster and consumes up to ≈40% less memory than *SES* while keeping the same good performance.**

***Index Terms*—Network Security, Scan Detection, Traffic Sampling**

I. Introduction

Traffic monitoring and analysis is essential for cybersecurity. Understanding what is really happening in a network is increasingly becoming more and more complex due to the ever-growing list of applications and the tremendous rise of cyberattacks and cybercrime worldwide [1]. Additionally, in high-speeds links this becomes even more challenging due to the common usage of sampled data as opposed to capturing full packets. Robustness against traffic sampling is now a topic of paramount importance since network operators tend to apply aggressive sampling rates when using monitoring tools like NetFlow [2] (e.g., take 1 packet out of 1000) in order to handle worst case scenarios and network cyberattacks. For this reason, it is fundamental to build sampling-resilient cyberthreat detection mechanisms.

In particular, we are interested in analyzing the performance of scan detection algorithms under sampling. We focus on network scanning for many reasons. Firstly, they are frequently the prequel of other cyberthreats (e.g., worm propagation) and, therefore, there is general interest in detecting them reliably. Secondly, scanning activities represent more than 80% of the cyberattacks on the Internet according to a recent study [3]. Moreover, scans can put monitoring platforms in serious trouble (the nature of this sort of cyberthreats can easily overflow flow tables due to the potentially large set of new flows generated by the scanner). Several methods for scan detection have been proposed in the literature. The straightforward approach flags a scanner when it connects to more than a certain number of destinations during a fixed interval of time. This method is implemented in both Snort [4] and Bro IDS [5]. The mechanisms analyzed in this paper (TRW [6] and TAPS [7]) are more complex and have shown to be more effective. Bro also implements TRW. The main idea behind TRW is that a scanner will fail more connections than a benign host, thus classifying a host as non-legitimate when it makes too many consecutive failed connections. TAPS is based on the observation that scanners visit many more destination IPs vs. ports than normal hosts (or the reverse, depending on the type of scan; vertical or horizontal). Likewise TRW, when this condition is accomplished several times, TAPS will report the host generating such flows as a scanner.

One of the main objectives of this paper is to present an independent validation on the impact of sampling on TRW and TAPS. Previous works (e.g., [8]) used the same percentage of sampled flows as the common metric to compare the different sampling methods, which resulted in biased conclusions reporting *Flow Sampling* as the best technique for anomaly detection under sampling. Instead, since every packet must be processed by the router, we proceed by taking the same fraction of packets. The motivation of this study came from the fact that given a flow sampling rate, the fraction of analyzed packets is significantly different among the sampling methods, which results in an unfair comparison, specially for *Packet Sampling*. First, we analyze the impact of sampling on TRW and TAPS under four sampling techniques (*Packet Sampling*, *Flow Sampling*, *Smart Sampling* [9] and *Selective Sampling* [10]) and confirm that both TRW and TAPS performance was vastly degraded due to sampling. While TRW reported quite poor results, TAPS showed to be more resilient. In contrast to the results reported by previous works on the poor performance of *Packet Sampling* for scan detection under sampling [11], [8], we reach significantly different conclusions. In particular, we show that when using the same fraction of packets, *Packet Sampling* outperforms *Flow Sampling*. Moreover, we show that a recently proposed sampling technique called *Selective Sampling*, which targets small flows normally used to perform scanning cyberattacks, exhibited the best overall performance among the evaluated sampling algorithms. Taking into account the good results reported by this technique, the second part of this paper proposes a new sampling method called *Online Selective Sampling* that also focuses on small flows but uses less resources because it works on a per-packet basis and, therefore, it can operate online.

In summary, we make the following contributions:

1) We perform an independent validation on the impact of sampling on TRW and TAPS.
2) Unlike previously reported, we show that *Packet Sampling* can perform better than *Flow Sampling*.
3) We present the first analysis of *Selective Sampling* for TRW and TAPS and show that it outperforms traditional sampling techniques.
4) We propose *Online Selective Sampling*, a sampling technique equivalent to *Selective Sampling* that operates on a per-packet basis without requiring to aggregate packets into flows to perform the sampling, thus using less memory and CPU time and being able to work online.

The rest of this paper is organized as follows. First, Section II reviews the related work and Section III describes the datasets, the network scenario and the followed methodology. Second, Section IV reports on the impact of sampling on scan detection. Afterwards, Section V describes and evaluates our sampling technique proposal, *Online Selective Sampling*. Finally, Section VI concludes this paper.

II. Related work

Few previous works have analyzed the impact of sampling on scan detection [8], [11], [10], [12]. In particular, the impact of *Packet Sampling* on TRW and TAPS was analyzed in [11]. It was observed that both the false positives and the false negatives increased significantly with sampling for TRW. It was also concluded that TAPS was more resilient to sampling than TRW because even though TRW had higher success ratio, TAPS exhibited a significantly lower ratio of false positives. The same authors extended the analysis to several sampling mechanisms in [8]. It was shown that *Packet Sampling* introduced an important bias in the flow size distribution and that techniques targeting large flows were not convenient for scan detection (scanning cyberattacks use small flows). It was concluded that *Flow Sampling* was the best choice for scan detection under sampling, while *Packet Sampling* was considered among the worst. This was a despairing result as routers only implement *Packet Sampling*. In [12] we presented a preliminary study of the performance of *Packet Sampling* using the same fraction of packets and showed that under some scenarios it could outperform *Flow Sampling*. Moreover, in [10], Androulidakis et al. show that opportunistic flow-based techniques that target a certain part of the traffic can improve the performance of cyberattack detection algorithms under sampling w.r.t. to the unsampled case or under random flow-based sampling techniques.

III. Scenario and Methodology

For the evaluation, we use four traffic traces from the Gigabit access link of our university, Universitat Politècnica de Catalunya BarcelonaTech (see Table I). This link connects about 10 campuses, 25 faculties and 40 departments to the Internet through the Spanish NREN (RedIRIS). The datasets are 60-minute traces from 2012 collected at different times of the day (morning, noon, evening and night).

TABLE I: Detailed information about the traces used.

Trace	Start Time	Duration	Flows	Packets	Bytes
dataset-1	03:00	60 min.	5.5M	59.5M	39.6G
dataset-2	09:00	60 min.	7.9M	128.3M	95.1G
dataset-3	15:00	60 min.	8.6M	136.2M	101.5G
dataset-4	21:00	60 min.	6.5M	94.3M	67.4G

A. Ground Truth

In order to analyze what is the impact of sampling on TRW and TAPS, we first need to establish a ground truth of true scanners. We followed the same approach proposed in [7], [8], [11], i.e., we created a super set of scanners. In our case, we run TRW, TAPS, Snort and Bro with loose parameters (refer to Section III-B for details). Afterwards, we used frequent item-set mining (FIM) and manual inspection as in [3] to further check that all the scanners were indeed malicious cyberattacks. FIM is a well-known data mining technique used to extract knowledge from the data by finding frequent correlations among elements. It is useful for anomaly detection because when an attack takes place, many traffic features with the same values appear together in a large amount of traffic flows. For example, when a vertical scan occurs, lots of traffic flows have the same source and destination IPs and, therefore, FIM easily identifies all of them.

After this process, we obtained the final list of true scanners (ground truth) against which we will compare the scanners detected after applying sampling. Note that although we can not guarantee that the obtained ground truth is purely composed by true scanners, this method increases the reliability of the ground truth w.r.t. previous works [7], [8], [11]. This is due to the usage of FIM, which automatically finds common patterns and, therefore, significantly reduces the need for human intervention, which is error-prone.

B. Methodology

We analyzed the following four sampling techniques: *Packet Sampling* (*PS*), *Flow Sampling* (*FS*), *Smart Sampling* (*SMS*) and *Selective Sampling* (*SES*). *PS* samples each packet randomly with probability p, $0 \leq p < 1$. Similarly, *FS* samples each flow also with a random probability p. *SMS* [9] always samples large flows and looks down on the small ones. A flow is considered to be large if its size is above a given parameter z. The probability of taking shorter flows is inversely proportional to their size. In contrast, *SES* [10] focuses on sampling small flows, which are normally used for launching scanning cyberattacks. It uses three different parameters: z, c and n. z corresponds to the threshold that defines the size of a small flow (in packets), c is the probability of sampling a small flow and n is used to further regulate the percentage of non-small flows taken.

We configured TRW and TAPS with a false positive ratio of 0.01, probability of detection to 0.99, probability of having a successful connection being a scanner to 0.2 and to 0.8 for a legitimate host as recommended in [6], [7]. As in [11], [8], the ratio used by TAPS to detect suspicious sources (k) and the time bin to check it (t) were configured differently for each traffic trace and sampling rate in order to obtain the optimal results. For further details about TRW and TAPS and their configuration parameters, refer to [6], [7]. For Snort we used the *sfPortscan* detection module with *scan_type=all* and *sense_level=low*, which ensures low false positives. Finally, for Bro's standard scan detection algorithm we used an alarm threshold of 25.

Even though it is not possible to configure a router to sample a certain percentage of the incoming flows, previous works have only considered a scenario where all sampling methods receive the same fraction of flows to perform the comparison among techniques [8]. However, most routers only support packet-based sampling (e.g., Sampled NetFlow). That is the reason why it is also important to perform the comparison with the same fraction of packets for all sampling techniques. Moreover, note that the sampling rate for packet-based (e.g., *PS*) and flow-based (e.g., *FS*) sampling techniques has different meanings. While in the first case it refers to the fraction of sampled packets, in the latter it indicates the portion of sampled flows. This results in a significantly different amount of sampled packets and flows among the different sampling methods. For instance, when sampling 10% of the flows from *dataset-3*, *PS* receives 3.01% of the packets, *FS* 9.2%, *SMS* 85.35% and *SES* 0.66%. When sampling 10% of the packets, *PS* receives 24.21% of the flows, *FS* 11.08%, *SMS* 0.0062% and *SES* 78.15%. Therefore, from the point of view of *PS*, the comparison was unfair in previous works because it was receiving less packets than *FS*. The reason why the same fraction of flows corresponds to such a small percentage of packets for *PS* is that the probability of sampling packets from the same flow is extremely low because most flows are small. Therefore, sampling more than one packet per flow is very unlikely.

Consequently, unlike previous works [8], we perform an alternative analysis using the same fraction of packets. Previous studies have analyzed instead the algorithms using the same fraction of flows. Note that one analysis is not better than the other but complementary. Our analysis complements previous results using a different perspective, which helps to better understand the behavior of these scan detection algorithms under sampling.

We analyze the performance of the algorithms using the following metrics previously defined in [7]: $SR = \#true\_scanners\_detected/\#true\_scanners$ and $FPR = \#false\_scanners\_detected/\#true\_scanners$. The success ratio (*SR*) indicates how efficient a particular algorithm under sampling is by computing what proportion of the detected scanners matches these scanners in the ground truth (*#true_scanners*). The false positive ratio (*FPR*) shows how correct is that algorithm, i.e., it reports the percentage of misclassified scanners (sources wrongly classified as scanners).

IV. Impact of Sampling on Scan Detection

In this section, we analyze the impact of the four sampling techniques presented in Section III-B on TRW (Section IV-A) and TAPS (Section IV-B) and discuss the obtained results (Section IV-C).

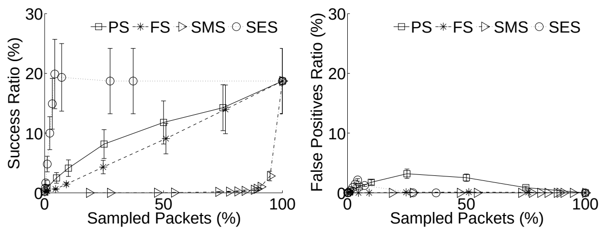

Fig. 1: Impact of sampling on TRW (mean ± stdev).

A. Impact of Sampling on TRW

Figure 1 reports on the performance of TRW under sampling. In particular, it shows the average and the standard deviation among all traces for the success ratio and the false positive ratio.

First of all, note that TRW is only able to detect $\approx 20\%$ of the scanners in the ground truth when there is no sampling, which highlights the fact that its low performance is mainly caused by the algorithm itself and not only because of sampling. Moreover, we confirm that sampling further degrades TRW's accuracy as previously reported [11], [8]. Regarding the sampling techniques, we can see that except for *SES*, the impact of sampling on TRW's success ratio (SR) is severe. For instance, under *SMS*, TRW's SR reaches zero with more than 75% of the packets sampled. This is due to the fact that, while TRW tracks single SYN-packet flows to spot scanners, *SMS* samples large flows, i.e., looks down on small flows and, therefore, keeps flows that are useless for TRW. For *PS* and *FS*, the SR degrades linearly for increasing sampling rates (s) even though *PS* is slightly better than *FS*. Note that previous works [8] reported that *PS* was worse than *FS*. However, the analysis was performed under the same fraction of flows, i.e., under unfair conditions for *PS* (see Section III-B). Therefore, this result shows that it is not true that *FS* is better than *PS* or vice versa. It essentially shows that the final conclusion depends on the metric you use to compare both methods. In contrast to the other sampling techniques, for *SES*, the SR shows to be equal to the unsampled case and gets even higher for s up to $\approx 1\%$. Moreover, for lower sampling rates, *SES* is still capable of detecting some scanners. The reason why TRW performs so good under *SES* is because this sampling method focuses on small flows, which are precisely these flows that TRW looks for in order to find scanners. *SES* outperforms the unsampled case because it drops non-small flows, and, therefore, leads TRW to a biased scenario where most of the hosts are only generating single SYN-packet flows.

Regarding the false positive ratio (FPR), all sampling techniques behave similarly, i.e., they all report low false positives. In particular, *PS* is the sampling technique performing the worst. Specifically, the highest peak ($\approx 3.2\%$) happens when sampling 25% of the packets. This is due to the well-known *flow-shortening* effect [11], [8], which transforms multi-packet flows into single packet flows and thus leads to the wrong classification of many hosts. Similarly, *SES* presents an unrealistic scenario where most of the hosts are only generating single SYN-packet flows. However, *SES* manages to keep a lower FPR for all sampling rates (its maximum value is close to

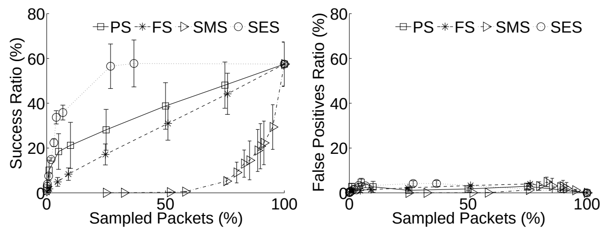

Fig. 2: Impact of sampling on TAPS (mean ± stdev).

2%). Both *FS* and *SMS* show almost no false positives because the former keeps the flow size distribution and the latter leads mainly to false negatives (low SR), but not to false positives due to the systematic drop of small flows.

B. Impact of Sampling on TAPS

Figure 2 reports on the performance of TAPS under sampling. Specifically, it shows the average and the standard deviation among all traces for the success ratio and the false positive ratio.

Firstly, note that, on average, TAPS is only able to detect 57.47% of the scanners in the ground truth when there is no sampling. Although the detection rate is far higher than TRW's, this highlights the importance of combining several mechanisms to detect a broader range of cyberthreats (e.g., in our case, using Snort, Bro, TRW and TAPS). The performance degradation among sampling methods is similar to TRW's. Specifically, we observe that *SES* is the sampling method offering the best success ratio followed by *PS* and *FS*, and finally, *SMS*. Similarly to the TRW case, the sampling methods focusing on small flows perform better than these that take large flows with higher probability. Moreover, we observe that like for TRW, *PS* reports higher SR than *FS*, which is not aligned with previous works [8]. Similarly to what we observed for TRW, TAPS also reports low false positives.

C. Discussion

It is clear that the performance of both TRW and TAPS is severely affected by sampling. However, TAPS is able to detect more scanners. The main reason behind such behavior is that TAPS does not depend on any specific packet. While TRW tracks single SYN-packet flows, TAPS does not care about what particular packet of a flow is taken but about the access patterns of each host. This makes TRW more sensitive to the particular packet being discarded. Moreover, while TAPS detects both UDP and TCP scans, TRW only detects TCP scans. However, although TAPS systematically reports higher SR, TRW shows slightly better FPR. Overall, as previously noticed [11], TAPS is preferable over TRW in the presence of sampling.

Regarding the sampling techniques, we reach two main conclusions. First, unlike previous works reported [11], [8], *PS* shows better results than *FS* when using the same fraction of packets for both TRW and TAPS (previous studies only used the same fraction of flows to compare them, which was unfair for *PS*). Second, *SES* clearly outperforms the other sampling techniques. For this reason and considering that most routers only support per-packet sampling, we propose a new packet-based implementation for *SES* (Section V).

V. Online Selective Sampling

Many cyberthreats such as scans use small flows to perform the attacks. Therefore, keeping these flows rather than the large ones facilitates identifying the anomalous traffic out of the whole traffic. There is a recent sampling proposal called *Selective Sampling* [10] (*SES*), whose goal is precisely to take just these small flows. The problem of *Selective Sampling* is that it first needs to capture all the packets and then samples entire flows, i.e., all incoming flows must be stored in memory until they expire. This working scheme is contradictory with the main goal of traffic sampling, which is precisely to save resources by discarding part of the traffic. Our sampling proposal, *Online Selective Sampling* (*OSES*), targets the same type of traffic but it does not need to capture entire flows because it takes per-packet decisions, thus requiring much less resources and being able to work online.

SES preferentially samples small flows (defined by a threshold in packets) and looks down on large flows. The key idea of our proposal, *OSES*, is to maintain a flow (i.e., to sample all its packets) while it is small. If we define a small flow as a x-packet flow ($x \geq 1$), *OSES* will sample all flows having at most x packets with a certain (high) probability and take all the flows having $x+1$ packets or more with lower probability (the more packets, the higher the discarding probability).

The straightforward solution used in [10] is to first store all flows in a hash table and then, as they expire, remove them from the table with a certain probability. However, this solution requires a lot of memory and is not fast enough in high-speed links. In particular, the inter-arrival times in links of several Gb/s are in the order of nanoseconds, thus requiring the process time per packet to be incredibly fast. In contrast, we base our solution on bloom filters, which are a feasible option due to their extremely quick look up time and low memory requirements. In our case a flow is kept in the hash table while it is not discarded by *OSES* (instead of waiting until it finishes or expires as in *SES*).

A. Our Proposal: Online Selective Sampling

Since *OSES* does not capture entire flows, we do not know their final size to decide whether they must be sampled or not. We only know the size of a flow until its current packet. Our goal is to be equivalent to *Selective Sampling* but without capturing all the traffic. Therefore, we want that the probability of taking a flow packet by packet is exactly the same that it would be when sampling the entire flow. If that occurs, both sampling methods will be equivalent. Consequently, *OSES* probability p must be adjusted at each step (for every packet) in such a way that the accumulated probability until the last packet results in exactly the same as directly sampling that flow with *SES*. In particular, the per-packet decisions must compensate for the higher number of random decisions taken w.r.t. the single decision required by *SES*.

Specifically, *OSES* works as follows. For each incoming packet we first check if its corresponding flow has been previously discarded by looking it up in the bloom filter. If there is a match, the packet is directly dropped because it means that the corresponding flow has been previously discarded. If there is no match, it means that the flow to which that packet belongs to, has not been dropped yet. In this case, the new packet is processed as follows. If the adjusted probability p is lower than a certain random probability, the new packet is dropped, its flow is deleted from the hash table and its 5-tuple is set in the bloom filter. Otherwise, the packet is sampled. Next, we explain how p is computed.

Consider that p_x^{ses} and p_x^{oses} are the probabilities of sampling a x-packet flow for *SES* and *OSES*, respectively. Our goal is:

$$p_x^{oses} = p_x^{ses} \tag{1}$$

While for *SES* this probability depends on a single random decision, for *OSES* it is the accumulation of x random and independent decisions, one for each packet of the flow:

$$p_x^{oses} = \prod_{i=1}^{x} r_i^{oses} \tag{2}$$

Consequently, from equations 1 and 2, the probability of sampling the x-th packet of a flow (r_x^{oses}) is:

$$p_x^{ses} = p_x^{oses} = \prod_{i=1}^{x} r_i^{oses} \rightarrow r_x^{oses} = \frac{p_x^{ses}}{\prod_{i=1}^{x-1} r_i^{oses}} = \frac{p_x^{ses}}{p_{x-1}^{oses}} \tag{3}$$

Recall the probability function of *SES* from [10] (refer to Section III-B for parameter details):

$$p_x^{ses} = \begin{cases} c & x \leq z \\ z/(n \cdot x) & x > z \end{cases} \tag{4}$$

In order to accomplish Eq. 1, p_x^{oses} must be equal to c up to z packets and to $z/(n \cdot x)$ when the flow is larger. Therefore, for a flow of $x-1$ packets:

$$p_{x-1}^{oses} = \begin{cases} c & x \leq z+1 \\ z/(n \cdot (x-1)) & x \geq z+2 \end{cases} \tag{5}$$

Next, we compute the probability of sampling the i-th packet for *OSES* (r_i^{oses}). First, note that for $x = 1$, $p_x^{oses} = r_x^{oses} = c$ to accomplish Eq. 1. For $x > 1$, from Eq. 3, Eq. 4 and Eq. 5, the individual probability of sampling the i-th packet of a flow is:

$$r_i^{oses} = \begin{cases} c & i = 1 \\ 1 & 1 < i \leq z \\ z/(c \cdot n \cdot i) & i = z+1 \\ (i-1)/i & i > z+1 \end{cases} \tag{6}$$

For instance, suppose that we have a flow of size $x = 5$ packets. We configure *SES* with $c = 0.9$, $n = 1$ and $z = 2$. According to *SES*' formula (Eq. 4) and these parameters, the probability of sampling that flow is $2/(1 \times 5)$ i.e., 40%. Using the same configuration for *OSES*, the sampling process works as follows: $p_5^{oses} = \prod_{i=1}^{i=5} r_i^{oses} = 0.9 \cdot 1 \cdot 0.74 \cdot 0.75 \cdot 0.8 = 0.4$. Indeed, we confirm that $p_5^{oses} = p_5^{ses}$.

TABLE II: Flow size distribution comparison between *SES* and *OSES* on *dataset-1* (c=0.9, z=2, n=1).

Method	1 pkt	2 pkts	3 pkts	4 pkts	5 pkts	6 pkts
SES	73.673%	12.893%	5.319%	3.250%	1.930%	0.99%
OSES	73.672%	12.891%	5.317%	3.241%	1.934%	0.997%

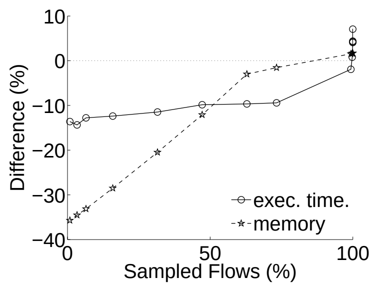

Fig. 3: Performance differences between *OSES* and *SES* in terms of both execution time and memory usage on *dataset-1*.

B. Validation

The first goal of this section is to show that the traffic sampled by *OSES* is equivalent to the traffic sampled by *SES*. The bloom filter was configured with $m = 2^{18}$ (the power of 2 is due to implementation reasons). We empirically observed a negligible number of false positives ($<< 1\%$ of the incoming packets). We compare the percentage of sampled flows and packets, the average flow size and the flow size distribution. The average and the standard deviation for several executions on *dataset-1* for $c = 0.9$, $z = 2$, $n = 1$ resulted in $76.53\% \pm 0.01$ of sampled flows, $11.98\% \pm 0.005$ of sampled packets and an average flow size of $1.68 \pm 8.94 \cdot 10^{-4}$ packets for *OSES*. Similarly, *SES* showed almost identical results: $76.54\% \pm 0.01$ sampled flows, $11.99\% \pm 0.01$ packets and an average of 1.68 ± 0.001 packets per flow. As we can observe, the differences among *SES* and *OSES* are negligible. These minor discrepancies are due to the random decisions. Moreover, Table II shows that the flow size distribution for both methods is almost identical. Note that for clarity reasons, only the percentage of flows from sizes 1 to 6 packets are reported (the remaining flow sizes account for less than 2% of the total flows all together). Therefore, after this analysis, we confirm that the traffic sampled by our proposal, *OSES*, is indeed equivalent to the traffic sampled by *SES* but without requiring to capture entire flows.

Figure 3 shows the resource consumption differences between *OSES* and *SES* on *dataset-1*. In particular, we can observe a percentage that indicates how *OSES* performs w.r.t. *SES*. For the execution time, the difference is computed as $(t_{OSES} - t_{SES})/t_{SES}$, where t is the execution time. Similarly, for the memory usage, the percentage is calculated as $(m_{OSES} - m_{SES})/m_{SES}$, where m is the maximum memory used. Therefore, while a positive percentage indicates that *OSES* was worse than *SES*, a negative value implies that *OSES* outperformed *SES*. As we can see in the figure, *OSES* is clearly better than *SES* in terms of both execution time and memory usage. For low sampling rates, *OSES* shows its best performance by using almost 40% less memory than *SES* due to the quick drop of a large amount of packets by the bloom filter. Also, *OSES* shows to be approximately 15% faster for the same reason. As the sampling rate gets higher, the differences between both methods are reduced but *OSES* still shows significantly better results. As the percentage of sampled flows gets closer to 100%, the overhead of the bloom filter in terms of both runtime and memory becomes visible up to the point that *OSES* turns out to be slightly worse (only for more than 99.8% of sampled flows).

The results obtained in this section accomplish our twofold objective, i.e., continue capturing scanners reliably under sampling and using less resources than *SES*. Moreover, while *SES* is not NetFlow-compatible because it is a flow-based sampling mechanism, *OSES* would be easily implementable to work online in most routers because, like NetFlow, it works on a per-packet basis.

Finally, we also evaluated the performance of TRW and TAPS under *OSES*. As expected, similarly to the results reported by *SES* in Section IV, *OSES* also showed very good results for scan detection under sampling.

VI. Conclusions

In this paper we have analyzed the impact of sampling on two scan detection algorithms to determine if they are robust enough to continue finding scanning cyberattacks reliably. In contrast to some previous studies, we have been able to see that, when treated fairly, *Packet Sampling* performed better than *Flow Sampling* for scan detection. Furthermore, we proposed a new sampling technique equivalent to *Selective Sampling* that keeps its good results for scan detection but uses less resources and is implementable in the widely deployed NetFlow.

Regarding the analyzed portscan detection algorithms, we confirmed that both TRW and TAPS were significantly impacted by sampling. However, even though TRW's performance was poor, TAPS showed higher resilience. Unlike previously reported, we found that *Packet Sampling* outperformed *Flow Sampling* when using the same fraction of packets as the common metric to compare techniques (previous studies had used the same fraction of flows). *Smart Sampling* showed to be rather useless for both TRW and TAPS. In contrast, *Selective Sampling* presented the best behaviour among all the sampling techniques achieving remarkable performance for both TRW and TAPS. Taking this into account, we proposed *Online Selective Sampling*, a new sampling technique similar to *Selective Sampling* that works on a per-packet basis instead of taking per-flow decisions, and, therefore uses less resources while keeping good performance for scan detection.

Acknowledgements

We thank UPCnet for the traffic traces provided for this study. This work was partially funded by the Spanish Ministry of Education under contract TEC2011-27474 and the Catalan Government under contract 2009SGR-1140.

References

[1] K.-K. R. Choo, "The cyber threat landscape: Challenges and future research directions," *Computers & Security*, vol. 30, no. 8, pp. 719–731, 2011.

[2] Cisco Systems, "Cisco IOS NetFlow," http://www.cisco.com/en/US/products/ps6601/prod_white_papers_list.html.

[3] M. Molina, I. Paredes-Oliva, W. Routly, and P. Barlet-Ros, "Operational experiences with anomaly detection," *Computers & Security*, vol. 31, no. 3, pp. 273 – 285, 2012.

[4] M. Roesch, "Snort–lightweight intrusion detection for networks," in *Proc. of LISA*, 1999.

[5] V. Paxson, "Bro: A system for detecting network intruders in real-time," *Computer networks*, vol. 31, no. 23-24, pp. 2435–2463, 1999.

[6] J. Jung, V. Paxson, A. Berger, and H. Balakrishnan, "Fast portscan detection using sequential hypothesis testing," in *Proc. of IEEE SP*, 2004.

[7] S. Avinash, T. Ye, and B. Supratik, "Connectionless portscan detection on the backbone," in *Proc. of IEEE IPCCC*, 2006.

[8] J. Mai, C. Chuah, A. Sridharan, T. Ye, and H. Zang, "Is sampled data sufficient for anomaly detection?" in *Proc. of ACM SIGCOMM IMC*, 2006.

[9] N. Duffield and C. Lund, "Predicting resource usage and estimation accuracy in an ip flow measurement collection infrastructure," in *Proc. of ACM SIGCOMM IMC*, 2003.

[10] G. Androulidakis, V. Chatzigiannakis, and S. Papavassiliou, "Network anomaly detection and classification via opportunistic sampling," *IEEE Network*, vol. 23, no. 1, pp. 6–12, 2009.

[11] J. Mai, A. Sridharan, C. Chuah, H. Zang, and T. Ye, "Impact of packet sampling on portscan detection," *IEEE JSAC*, vol. 24, no. 12, pp. 2285–2298, 2006.

[12] I. Paredes-Oliva, P. Barlet-Ros, and J. Solé-Pareta, "Portscan detection with sampled netflow," in *Proc. of TMA*, 2009.

Ignasi Paredes-Oliva *received his B.Sc. and M.Sc. degrees in Computer Science in 2009 from Universitat Politècnica de Catalunya BarcelonaTech (UPC). He is a PhD candidate at UPC and his research interests are on current challenges for network cybersecurity in backbone networks such as anomaly analysis, anomaly detection or anomaly classification. Contact him at iparedes@ac.upc.edu.*

Pere Barlet-Ros *received his M.Sc. and Ph.D. degrees in Computer Science in 2003 and 2008 from Universitat Politècnica de Catalunya BarcelonaTech (UPC). He is an assistant professor and researcher at UPC and his research interests are in the fields of network monitoring, traffic classification and anomaly detection. Contact him at pbarlet@ac.upc.edu.*

Josep Solé-Pareta *obtained his M.Sc. degree in Telecom Engineering and Ph.D. in Computer Science in 1984 and 1991 from Universitat Politècnica de Catalunya BarcelonaTech (UPC). He is a full professor at UPC and his research interests are in Nanonetworking Communications, Traffic Monitoring and Analysis and High Speed and Optical Networking. Contact him at pareta@ac.upc.edu.*