Analysis of the Impact of Traffic Sampling on Portscan Detection

Ignasi Paredes Oliva - iparedes@ac.upc.edu
Pere Barlet Ros - pbarlet@ac.upc.edu
Josep Solé Pareta - pareta@ac.upc.edu

UPC Technical Report

Departament d'Arquitectura Computadors
Universitat Politècnica de Catalunya

Abstract

Although anomaly detection has been widely studied in the literature, recently, another related topic came into scene being object of high interest: the impact of sampling. Due to current high-speed links is very hard to process all the incoming packets and using a sampling technique (discard a portion of the traffic) has become almost mandatory.

There are lots of anomaly detection mechanisms made to detect distinct kinds of attacks. The question is if all these already existing solutions will work in the presence of sampling. How much resilient to sampling these mechanisms will be? Will they be capable of continue detecting anomalies in a reliable way?

This technical report focuses on one particular anomaly: portscans. Our choice is mainly motivated by the fact that it is one of the most common attacks together with DoS/DDoS and it usually precedes other attacks such as worm propagation. We have studied the impact of sampling on two well-known portscan detection algorithms of the state of the art: TRW and TAPS. We analyzed five different sampling techniques: *Packet Sampling*, *Flow Sampling*, *Sample and Hold*, *Smart Sampling* and *Selective Sampling*.

There are a few studies dealing with the impact of sampling on portscan detection that concluded that the best sampling technique for anomaly detection was *Flow Sampling*. Since we are basically interested in NetFlow (protocol used to aggregate packets into flows), and *Flow Sampling* is a flow-based sampling method (not implementable by NetFlow), we started looking for the reasons why *Packet Sampling* did not perform well for anomaly detection. We realized that it was not fairly treated in previous works. For example, according to one of our traces, when sampling 10% of the flows, we realized that while *Packet Sampling* did not even take 3% of the packets, *Flow Sampling* sampled $\approx$ 11% of them and the other sampling techniques even keeped a larger proportion of the traffic. Therefore, we changed that conditions and performed our own experiments proceeding in two different ways: the already used procedure plus our own, which treated *Packet Sampling* fairly.

We did our evaluation using SMARTxAC, a passive network monitoring system analyzing the link between the *Anella Científica* (it connects many universities and

research centers in Catalonia) and the Spanish National Research and Educational Network (RedIris). Because of our new way to proceed we found a new conclusion not entire aligned with previous studies. We have been able to empirically check that *Flow Sampling* is not always better than *Packet Sampling* for portscan detection. Regarding the anaylzed portscan detection algorithms, they both were dramatically impacted by sampling. While TRW showed to be not quite useful under sampling, TAPS exhibited more acceptable performance. Concerning to the sampling techniques, using a NetFlow compatible sampling technique (packet-based sampling), *Packet Sampling* turned out to be the best. Without NetFlow, *Selective Sampling* achieved significantly higher performance than the rest.

Finally, we propose a new packet-based sampling technique, called *Online Selective Sampling* that focuses on small flows, which usually are the source of many attacks like DDoS and portscans. It is equivalent to an already existing sampling technique called *Selective Sampling*, which is also analyzed in this work. Our method, instead of requiring aggregation of packets into flows before sampling, works on a packet-by-packet basis, thus not capturing all the traffic and using significantly less memory. Furthermore, since it takes per-packet decisions it is implementable in NetFlow. However, our method still requires further validation and a more complete analysis.

Chapter 1

Introduction

Traffic monitoring and analysis is essential for security and management tasks. In high-speeds links it is difficult to process all the incoming packets because nowadays, the inter arrival times among packets are incredibly small. For instance, in links of several Gps, we are talking about the order of nanoseconds between packets, which means that the software and the hardware in charge of receiving and processing the incoming stream of packets must be really quick. Furthermore, attacks may happen, which implies that routers could be even under more stress due to abnormal traffic bursts.

In order to alleviate so much load and avoid possible saturations because of anomalies, it turns out that discarding some of the incoming traffic (sampling) is nearly compulsory. Robustness against sampling is very important since network operators tend to apply aggressive sampling rates when using NetFlow (e.g., take 1 packet out of 1000) in order to handle worst case scenarios and attacks. For this reason, it is fundamental to build sampling-resilient anomaly detection mechanisms. NetFlow [1] is a protocol used to export IP information from routers. It aggregates packets into flows (usually defined by the source and destinations IPs and ports together with the protocol) and then, it expires them every certain interval of time. There are lots of different sampling techniques that sample traffic according to some particular criterion. For instance, NetFlow has its own sampling technique called Sampled NetFlow [2], which samples on a per packet basis.

This technical report focuses on portscan detection due to two main reasons. Firstly, they are one of the most common attacks (e.g., they usually precede worm propagation) and, therefore, there is general interest in detecting them reliably. Secondly, portscan attacks can put NetFlow-based monitoring platforms in serious trouble (the nature of this sort of anomalies can overflow flow tables due to the potentially large set of new flows generated by a scanner). Several methods for portscan detection exist. The most basic one flags a scanner when it connects to more than a certain number of destinations during a fixed interval of time.

This one is the portscan detection algorithm implemented by the Snort IDS [3]. The portscan detection mechanisms tested in this study (TRW [4] and TAPS [5]) are more complex and have shown to be more effective. In particular, TRW is implemented in the Bro IDS [6]. The idea of TRW is that scanners will fail far 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 (or the reverse) than normal hosts do. Like with TRW, when this phenomenon happens several times, the host generating such flows is classified as a scanner.

Few recent studies have analyzed the impact of sampling on anomaly detection [7, 8, 9]. Mai et al. studied the impact of packet sampling on TRW and TAPS in [7]. In the case of TRW, they observed that false positives and negatives increase with sampling. They also concluded that TAPS is more resilient to sampling than TRW because even that TRW had better success ratio, TAPS exhibited a lower ratio of false positives. In [8], they compared several sampling mechanisms. They concluded that flow sampling was the best choice for anomaly detection under sampling. Finally, in [9], Brauckhoff et al. concluded that entropy summarization is more resilient to sampling than volume-based metrics.

In this work, we analyze the impact of sampling on two widely known portscan detection techniques in the literature: TRW and TAPS. In particular, we evaluated them under five sampling techniques: *Packet Sampling*, *Flow Sampling*, *Sample and Hold*, *Smart Sampling* and *Selective Sampling*. Except for *Selective Sampling*, the other sampling techniques were chosen because they are widely known and used in the state of the art, specially *Packet Sampling* (e.g., Sampled NetFlow is very extended among network operators, and it is based on *Packet Sampling*). The reason why we decided to analyze the former sampling technique as well, was because we saw that it targeted exactly the kind of flows that are responsible of portscans (small flows), thus being highly interested in how it could work.

One of the main objectives of this paper is to validate previous results in our network scenario when using Sampled NetFlow data. We also aim to evaluate the impact of the different sampling techniques on two portscan detection methods taking the same fraction of packets, while previous works (e.g., [8]) used instead the portion of sampled flows as the common metric to compare the different sampling methods. Although the amount of memory used by NetFlow to keep the flow tables is directly proportional to the number of flows, we focused on another relevant resource: the CPU cycles. Since in NetFlow every packet must be processed, it is also important to compare the accuracy of all sampling methods according to the ratio of sampled 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*. For instance, according to one of our traces, sampling 10% of

flows results only in 2.86% of sampled packets for *Packet Sampling*, while *Flow Sampling* gets 10.90% and *Sample and Hold* and *Smart Sampling* take even a larger proportion of packets (15.58% and ≈85% respectively).

Our first contribution is to validate the impact of sampling on two well-known portscan detection algorithms: TRW and TAPS. Furthermore, in addition to do the experiments using the same fraction of flows as already did by previous works, we proceeded keeping the same percentage of packets in order to see how *Packet Sampling* performed under fair conditions. This allowed us to find that *Flow Sampling* is not always better than *Packet Sampling* for portscan detection. Regarding the portscan detection algorithms, both TRW and TAPS performance was vastly degradated due to sampling. While TRW turned out to be almost useless under sampling (except using it under *Selective Sampling*, in which case it exhibited a quite acceptable performance), TAPS showed to be significantly more resilient. Concerning to sampling techniques, *Packet Sampling* showed to be the best to use under NetFlow. Without taking NetFlow into account, *Selective Sampling* exhibited the best performance among all sampling methods.

We finally proposed a new sampling method called *Online Selective Sampling* that targets small flows, which normally are used to perform portscan among other attacks like DoS/DDoS. The idea is to sample the same flows that Selective Sampling, but, instead of collecting all the packets in the first place and then discarding entire flows, it decides to sample on a per-packet basis. When the size of a flow in packets is higher than the threshold, the flow is discarded with a probability directly related to its size (the more packets the higher is that probability). While the flow is small, it is sampled according to a given probability. Preliminary results showed that *Online Selective Sampling* samples approximately the same fraction of the traffic that *Selective Sampling* but without capturing all the traffic, thus saving a lot of memory. Nevertheless, further analysis and validation about the method are still pending.

The rest of this technical report is organised as follows. In Chapter 2, the state of the art of anomaly detection is widely explained. Chapter 3 presents the analyzed sampling methods together with the evaluated portscan detection algorithms. Then, in Section 4, we describe our network scenario and the followed methodology as well as the data traces used. Section 5 shows and discusses the obtained results using real-world NetFlow data from a large university network. Our new sampling technique is proposed and deeply explained in Chapter 6. Finally, Section 7 concludes this study and summarises our current and future work.

Chapter 2

Related work

The amount of research related to anomaly detection is huge and still increasing every day because of the incredibly expansion of Internet. The already existing threats change fast and every day new and unknown anomalies show up. In addition to that, sampling techniques like Sampled NetFlow [**?**] are needed in core networks in order to avoid network resources saturation due to current high-speeds. Lots of research groups around the world try to deal with this rapidly changing environment building new anomaly detection mechanisms and sampling techniques that make it possible to continue detecting anomalies reliably taking into account the before described factors.

2.1 Anomaly detection

The state of the art of anomaly detection could be splitted into two general groups: specific-anomaly detection and general-purpose anomaly detection methods. The first set includes particular mechanisms to detect concrete anomalies while the second one provides mechanisms capable of detecting anything that differs from usual, which means that mechanisms in that group can detect both known and unknown anomalies. The non-specific solutions group could be divided again into two new subgroups: detection and identification. In the detection phase you find out that something was wrong but you do not know what it was (e.g., it was a DoS or a scan?). The identification part is in charge of determining the high-level anomaly that corresponds with the several symptoms detected before.

In [10], Barford et al. classified anomalies into three groups: network operational (e.g., configuration changes), flash crowds (sudden increase of general interest in some particular website) and network abuses (e.g., DoS). Then, they described how to detect each kind of these anomalies. The first group showed distinct levels of bit rate, while the second one exhibited a change in packet rate

and the last presented a significant increase of flows per time unit. In order to characterise precisely each group they did time series analysis (to provide predictive capability) and wavelet analysis (to gain insight into the structure of each anomaly type).

2.1.1 General-purpose anomaly detection mechanisms

The proposed solutions can be basically separated into these groups: threshold-based [11, 12], profile-based [13, 14], subspace-based [15, 16, 17, 18], wavelet-based [19] and entropy-based [18, 20].

Threshold-based anomaly detection mechanisms consider different periods of time: weekdays, weekends, day, night, etc. For each combination of these intervals, two thresholds are defined: the upper one and the lower. When some specific measured metric (e.g., #pkts, #flows, #bytes...) exceeds its corresponding upper or lower value an anomaly is reported.

Profile-based mechanisms use prediction to know what is the next expected value (time series prediction). If the measured metric differs too much from its prediction, it is considered to be an anomaly. There are several approaches to do that. *Exponential smoothing* [21] is the simplest method for prediction: the predicted value is extracted from the average between the current prediction and the current real value. The problem of this approach is that it does not account for seasonality, so it would always report anomalies when there were fast changes due to normal activity. The other prediction model is called *The Holt-Winters Forecasting Algorithm* [21] and it tries to overcome the previous problem. Its prediction is an average of three variables that account for baseline, linear trend and seasonality respectively. Each of these variables is updated using exponential smoothing.

The subspace-method [17] analyses what the authors call OD-flows (flows with the same Origin and Destination points of the monitored network). Because of the high dimensional multivariate data structure of that flows, a lower-approximation is needed: Principal Component Analysis (PCA) [16]. This mathematical method captures the most important trends of the explored data (it preserves the significance of the data while reducing its complex initial structure). Then, the subspace method divides the resulting set into normal and anomalous. If the projection of the data in the second space is higher than a previously given value, an anomaly is flagged. All that procedure is known as (single-way) subspace method but there are other mechanisms based on that one like the multi-way subspace method [18]. There, Lakhina et al., focused on data distribution: they wanted to detect the anomalies according to changes on the distribution of some particular metrics. The way they studied the distribution of a certain data set is the entropy: the higher is the dispersion of the data the higher is the entropy value (a 0 value of

entropy means no variation). They claim that entropy allows to detect unseen anomalies by volume metrics (e.g., number of packets or bytes). First, they compute the anomaly for each OD-flow and feature (sources and destinations IPs and ports) and then, they apply the before explained subspace method.

Next mechanisms are wavelet-based. A wavelet transform is the procedure of dividing a given signal into different frequency components. Applied to anomaly detection it is used in such a way that each component is used to look for anomalies matching its scale: low-frequency components contain very sparsed anomalies, so they are used to detect long-time anomalies. High-frequency components are useful to detect spontaneous changes (short-term anomalies).

Concerning to entropy-based mechanisms, in addition to [18], there are other different approaches like [20]. In that paper, Xu et al, detect anomalies according to changes on the distribution of some specific measured metrics. The way they studied the distribution of a certain data set is the entropy: the more disperse the data is, the higher is its entropy value. First of all, the distribution probability is calculated for each metric: source IP, destination IP, source port and destination port. After that, the most significant values are selected from each set. For each of these values they observed how the other three related metrics changed (e.g., if testing a certain source IP, they investigated what happened with the destination IP, source port and destination port for all flows having that source IP). According to that three values they made their decision (e.g., a scanner IP would have a large accessed set of destination IPs or ports with fixed or random source ports).

Regarding identification mechanisms, there are several approaches: subspace-based, cluster-based... In the first case, the idea is that somehow you have all the known anomalies characterised mathematically. Then, each anomaly in that set is compared against the direction of the vector that they obtained projecting the data onto the anomalous subspace (as explained before in this section). Then, the high-level anomaly that explains the largest amount of residual traffic is selected. The second solution works with clustering. It does not require supervision because it is capable of grouping together anomalies with similar behaviour. There are several algorithms like *k-means* and *agglomerative hierarchical* [18]. Although it is automatic, it requires an additional manual step because each final cluster might contain more than one high-level anomaly. Thus, the mapping between the obtained clusters and the true high-level anomaly (flash crowd, DoS, scan, ingress-shift...) could require human intervention.

2.1.2 Specific anomaly detection mechanisms

Portscan Attacks

In [22], Lee et al. described the distinct scanning attacks and also defined their general behaviour. They differentiate three kind of scans: vertical, horizontal and some kind of mixture between these two. First scan type means that the scanner looks for open ports on a single target destination IP, while the second one means that it checks for one specific open port on several target machines. The "mix" scan type refers to an scanner checking a fixed range of ports on a specific set of destination machines. There are lots of proposed solutions for detecting portscans. In this technical report, we will focus on those ones integrated in widely used Intrusion Detection Systems like Snort [3] and Bro [6]. The most basic detection mechanism just keeps a counter of the number of contacted destination ports and IPs from a given source IP. There are other solutions that are capable of achieving higher detection rates: Threshold Random Walk (TRW) [4] and Time based Access Pattern Sequential hypothesis testing (TAPS) [5]). These last two techniques are widely explained and analyzed in the following chapters.

DoS/DDoS Attacks

[23] provides some kind of survey about the existing DoS attacks and their respective proposed detection mechanisms. The idea of the authors is to establish a common framework among researchers, to detect weaknesses and compare different detection mechanisms.

The main goal of the majority of DoS related papers is to detect the attack while others like the proposed in [24] is to take care of "good" clients that are drowned out while an attack of this kind is being performed. The basic problem is the following: when there is a DoS, the good clients cannot be attended because the server cannot handle their requests due to the huge demand coming from the "bad" clients. The idea is to ask to the clients to increase their bandwidth usage to the maximum they can so that the fraction of the aggregation of good clients requests on the server is higher than the fraction coming from the attacker. If that is accomplished, legitimate hosts should receive the correct service.

2.2 Impact of Sampling on Anomaly Detection

There is not a lot of research dealing with the impact of sampling for anomaly detection but some work had been already done: [8, 9, 7, 25, 26, 27]. All these proposals check already existing anomaly detection mechanisms testing them under different sampling rates and distinct methods.

Mai et al. studied the impact of packet sampling on portscan detection in [7]. TRW, TAPS and an entropy-based anomaly detection technique were tested. Concerning TRW, they found out that flow size became lower in the presence of sampling, thus resulting in more false positives and a significant increase of the success ratio. They also showed that the metric used by TAPS (relation between number of accessed destination IPs and ports) is less affected, thus concluding that TAPS is better than TRW under sampling. When comparing both mechanisms under sampling, they observed that, while TRW had better success ratio, TAPS exhibited a lower ratio of false positives. Regarding to the entropy-based analysis, it presented a performance very similar to TAPS mechanism due to the fact that both mechanisms detect scanners profiling destination access patterns. They even proposed a new portscan detection method based on TAPS called TAPS-SYN, that reduced the false positives ratio.

In [8], the same authors tested several sampling methods (*Packet Sampling*, *Flow sampling*, *Sample-and-Hold* and *Smart Sampling*) against the two portscan detection mechanisms analyzed in this technical report (TRW and TAPS) and a wavelet-based volume anomaly detection mechanism. They found out that *Flow Sampling* performed better for both portscan and volume anomalies detection, while the other methods obtained very poor results. While *Packet Sampling* suffered the well-known flow shortening, *Sample and Hold* and *Smart Sampling* showed to be not suitable for anomaly detection since they look down on small flows (and attacks like portscans and DoS are made using that kind of flows).

In [9], Brauckhoff et al. studied how specific metrics are affected by sampling looking at counts of bytes, packets and flows, together with several feature entropy metrics. They concluded that entropy summarization is more resilient to sampling than volume-based metrics.

In [25], the impact of sampling on a Change-Point Detection (CPD) method [14] and a PCA-based anomaly detection mechanism [28], is analyzed. They made the experiments under several sampling methods proposed in the PSAMP IETF draft [29]: *Systematic sampling*, *Random n-out-of-N Sampling* and *Uniform Probabilistic Sampling*. They concluded that *Sistematic Sampling* is almost useless for anomaly detection when the anomaly detection mechanism relies on specific packet characteristics (e.g., TCP flags). When flow-based metrics are used instead, the performance is directly related to the sampling rate. Regarding the PCA-based method, the same behaviour is observed (it does not care about the particular sampling method used).

In [27], Androulidakis et al. designed a new flow-based sampling technique called *Selective Sampling* that focuses on small flows. These flows are usually responsible of port scanning and DoS activity. Then, they tested how it worked under a CPD anomaly detection mechanism. Moreover, its performance is compared with another flow-based sampling method (*Smart Sampling*) and against *Random*

Flow Sampling. This new sampling technique showed to improve the anomaly detection effectiveness and, sometimes, it even outperformed the unsampled case. Unlike in previous works [7], they observed that *Smart Sampling* overcame *Flow Sampling* in some cases, thus concluding that it can be used for anomaly detection if the used detection mechanism relies on large flows.

In [26], the authors extended their previous work testing their new sampling technique against a PCA-based anomaly detection mechanism. They observed that *Selective Sampling* exposed the anomalies more clearly than *Flow Sampling* did, and, in some cases, even better than the unsampled case. They attributed this improvement to the fact that the nine metrics that they used for PCA were more correlated in small-flows (*Selective Sampling* focuses on small flows).

Chapter 3

Background

3.1 Sampled NetFlow

NetFlow [1] is a network protocol developed by *Cisco Systems* used to collect IP traffic information on routers. It basically aggregates incoming packets into flows (NetFlow records) and expires them every certain interval of time. The flow definition is not exactly the same among the different NetFlow versions but it is mainly composed by the source and destination IPs, the source and destination ports and the protocol. As we can deduce from the definition a flow is considered to be unidirectional since a response flow would swap the the IPs and the router would find no match with any previous flow, thus creating a new one. In order to avoid resource exhaustion on routers due to network anomalies or bursts in the traffic NetFlow allows sampling. When *Sampled Netflow* [2] is turned on, the router samples one packet out of N for every interval.

3.2 Analyzed Sampling techniques

IETF Packet Sampling (PSAMP) working group [29] goal is to define standard ways to sample subsets of packets in some particular way. They recently (March 2009) released the RFC 5475 entitled *Sampling and Filtering Techniques for IP Packet Selection* [30] where several sampling techniques are explained in detail.

We have analyzed *Packet Sampling* (*PS*), *Flow Sampling* (*FS*), *Sample and Hold* (*SH*), *Smart Sampling* (*SMS*) and *Selective Sampling* (*SES*). While the first four sampling techniques are widely known in the literature and often used, we decided to add the fifth sampling mechanism due to to specific reasons directly related to the traffic profile which we are interested in for portscan detection under sampling (explained later).

PS is widely used because of its low CPU consumption and memory require-

ments. Flow-based approaches (e.g., *SH*) overcome some of the shortcomings of *PS* (it does not keep the original flow distribution) but, in exchange, they have higher resource requirements. Thus, some trade-off between accuracy and resource requirements is needed. Next, we present a brief explanation for each analyzed sampling technique.

3.2.1 Random Packet sampling

Random packet sampling takes each packet with probability p. There are several versions [30] of this mechanism such as *Systematic Packet Sampling* and *Random Packet Sampling*. The first one samples packets in a deterministic way (take every k-th packet), while the second one depends on random decisions. Our *PS* implementation corresponds to this last one. *Sampled NetFlow* is based on *Packet Sampling*, since it samples traffic in a per-packet basis.

3.2.2 Random Flow sampling

Random flow sampling takes each flow with probability q. This technique is usually implemented hashing the flow ID (e.g., the 5-tuple formed by the source and the destination IP addresses and ports, and protocol field). The flow is then selected if the resulting value (mapped to the [0..1) range) is below q [31].

3.2.3 Sample and Hold

Sample and Hold takes the packet directly if its flow ID belongs to an already seen flow. Otherwise, the packet is sampled with probability r. r is computed as $1-(1-h)^s \approx h \cdot s$, where s is the size of the packet and h is the probability of sampling a single byte [32].

3.2.4 Smart Sampling

Smart Sampling is a sampling technique that focuses on large flows and drops the small ones. In order to define what large means a threshold is provided to the algorithm. This value (z) indicates the size of the flows (in bytes) that you want to keep. All flows above that threshold will be sampled and the other ones will be discarded with a certain probability $p(x)$ related to their size (x):

$$p(x) = \begin{cases} x/z & x < z \\ 1 & x \geq z \end{cases}$$

SMS needs to capture all the packets in the first place. After the aggregation of packets into flows it can proceed with its flow-based sampling by sampling or discarding entire flows [33].

3.2.5 Selective Sampling

SES concentrates on small flows, which typically precede worm propagation and are used for DoS/portscan attacks. It works using three different parameters: z, c and n. The first one corresponds to the threshold that defines a small flow size (in packets). c is the probability of sampling an small flow (a flow with z packets at most). Finally, the parameter n is used to further regulate the percentage of non-small flows taken. A flow of size x packets is sampled/discarded according to the following expression:

$$p(x) = \begin{cases} c & x \leq z \\ z/(n \cdot x) & x > z \end{cases},$$

z,c and n must fulfil the following constraints: $z \geq 1$, $c > 0$ and $n \geq 1$. As *Smart Sampling* does, *Selective Sampling* must take all the packets before starting the sampling process [27].

3.3 Portscan Detection mechanisms

Simple portscan detection algorithms, like the one used by the Snort IDS, are useless nowadays since attackers can easily evade detection by reducing their scanning rate. There are many other techniques capable of achieving higher rates of detection such as TRW and TAPS, which we analyse in this technical report. We next provide a short explanation of each portscan detection mechanism tested.

3.3.1 Threshold Random Walk

The main idea behind Threshold Random Walk (TRW) [4] is that one scanner will fail more connections than a legitimate client when trying to establish a connection. The method works under two hypothesis: the host is either legitimate (H_0) or scanner (H_1).

$$Y_i = \begin{cases} 0 & \text{if connection successful} \\ 1 & \text{if connection failed} \end{cases}$$

$$Pr[Y_i = 0|H_0] = \theta_0 \qquad Pr[Y_i = 1|H_0] = 1 - \theta_0$$
$$Pr[Y_i = 0|H_1] = \theta_1 \qquad Pr[Y_i = 1|H_1] = 1 - \theta_1$$

Due to the fact that scanners will fail more connections than benign hosts, the following condition must be accomplished:

$$\theta_0 > \theta_1$$

Since it is possible to fail some connections even being a benign host, the decision of flagging a host as a scanner is not taken just after the first failure. For each source there is an accumulated ratio that is updated each time a flow ends.

$$\wedge(Y) = \Pi_{i=1}^{n} \frac{Pr[Y_i|H_1]}{Pr[Y_i|H_0]}$$

The update is done accordingly to the flow state: connection established or failed attempt. We did our experiments with an unidirectional trace, so we used the proposed modification of TRW, called *TRWSYN* [5], that identifies a failed connection when an ended flow is a single SYN-packet. Eventually, if any source IP keeps scanning, it will fail more and more connections and finally it will exceed the established threshold, thus being recognised as a scanner. Similarly, a legitimate host making several successful connections will be classified as non-scanner.

3.3.2 Time-based Access Pattern Sequential hypothesis testing

Time-based Access Pattern Sequential hypothesis testing (TAPS) [5] is based on the observation that the ratio between the number of destination IPs and the number of destination ports (or the reverse) when the source IP is an scanner is significantly higher than the same ratio when there is no scanning activity.

As the name suggest, TAPS is based on the same statistical method that TRW uses. It works almost the same way. It changes two things: the way it detects a suspicious behaviour and the update mechanism . It checks the following fractions every time bin (which is a configurable parameter given to the algorithm):

$$\frac{\#accessed\_IPs}{\#accessed\_ports} > k \qquad\qquad \frac{\#accessed\_ports}{\#accessed\_IPs} > k$$

The left side expression indicates a vertical scan, which indicates that the attacker is looking for a particular service (port) in a set of machines. If the reverse fraction is violated (right side formula), it means that a horizontal attack is going on, which tells us that either a particular machine or a small set of machines is/are being deeply scanned in order to see what ports it/they has/have opened.

When any of these two fractions is higher than a pre-configured threshold, the per-source IP ratio is updated. When this accumulated value reaches a certain limit, that source is classified either as a scanner (upper threshold) or a legitimate host (lower threshold).

Chapter 4

Scenario and methodology

4.1 Scenario

We collected five NetFlow traffic trace from the Gigabit access link of the Universitat Politècnica de Catalunya (UPC) (see Table 4.1 for more detailed information). This link connects about 10 campuses, 25 faculties and 40 departments to the Internet through the Spanish National Research and Education Network (RedIRIS). Real-time statistics about the traffic of this link are available on-line at [34].

4.2 Methodology

We first implemented the portscan detection techniques and the sampling methods described in Chapter 3 on the SMARTxAC monitoring system [12]. Then, we ran several tests with varying sampling rates, sampling methods and portscan detection algorithms. In order to have a ground of truth to compare with, we first ran each portscan detection algorithm without sampling (see Chapter 5 for more details about the used ground truth). After that, we can compare which attacks were missed in each case. We used the following sampling intervals $N = \{1, 10, 50, 100, 500, 1000\}$ to do our experiments.

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 by [4, 5]. After some tests, we fixed the ratio used by TAPS to detect suspicious sources to $Z = 3$. We fixed the time bin that TAPS uses to check source IP's ratios to 300 seconds.

Table 4.1: Detailed information about the NetFlow traces used

Trace	Date	Start time	Duration	Packets	Bytes	Flows
UPC-I	06-11-2007	16:30	30min.	105.38×10^6	61.86×10^9	5.26×10^6
UPC-II	21-12-2008	12:00	1h.	74.48×10^6	33.37×10^9	3.79×10^6
UPC-III	21-12-2008	16:00	1h.	71×10^6	31.40×10^9	4.33×10^6
UPC-IV	22-12-2008	12:30	1h.	163.51×10^6	115.28×10^9	6.44×10^6
UPC-V	21-12-2008	19:00	1h.	132.79×10^6	77.05×10^9	5.07×10^6

4.3 Computing sampling rates

It is important to note that the sampling rate in the case of *PS* and flow-based sampling techniques has different meanings. While in the first case it refers to the fraction of sampled packets, in the latter case it indicates the portion of sampled flows. This results in a very different number of sampled packets and flows among the different sampling methods. In order to make all the sampling methods comparable, we used the following two metrics:

4.3.1 Equal portion of packets

We first computed the packet sampling rate as $1/N$ for *PS*. Given this fraction of packets to keep, we then performed several tests to find the suitable sampling rates for the other sampling techniques in order to select the same portion of packets.

4.3.2 Equal portion of flows

We calculated the flow sampling rate as $1/N$ for *FS*. Given the portion of flows to take, we ran various tests to obtain the correct sampling rate values for *PS*, *SH*, *SMS* and *SES* in order to sample the same portion of flows.

4.3.3 Example for UPC-I trace

Tables 4.2 and 4.3 present the selected sampling rates that assure that the same portion of packets or flows is selected for all the sampling methods (using UPC-I trace).

Table 4.2: Percentage of taken flows given a portion of sampled packets for trace UPC-I

N	%pkts	*PS*		*FS*		*SH*		*SMS*	
		p	%flows	p	%flows	h	%flows	z	%flows
10	10%	0.1	25.89%	0.092	10.24%	1.06×10^{-4}	6.84%	60×10^6	0.019758%
50	2%	0.02	7.95%	0.026	2.78%	2.8×10^{-5}	2.03%	320×10^6	0.004122%
100	1%	0.01	4.70%	0.015	1.85%	1.5×10^{-5}	1.05%	750×10^6	0.001766%
500	0.2%	0.002	1.44%	0.0036	0.95%	4×10^{-6}	0.53%	1000×10^6	0.001114%
1000	0.1%	0.001	0.88%	0.0018	0.77%	2.7×10^{-6}	0.49%	1900×10^6	0.000611%

Table 4.3: Percentage of taken packets given a portion of sampled flows for trace UPC-I

N	%flows	*PS*		*FS*		*SH*		*SMS*	
		p	%pkts	p	%pkts	h	%pkts	z	%pkts
10	10%	0.028	2.86%	0.1	10.90%	1.8×10^{-4}	15.58%	12.5×10^3	85.83%
50	2%	0.003	0.33%	0.02	1.60%	2.8×10^{-5}	1.98%	14×10^4	73.32%
100	1%	1.2×10^{-3}	0.12%	0.01	0.59%	1.5×10^{-5}	1.05%	4×10^5	63.02%
500	0.2%	1.3×10^{-4}	0.01%	0.002	0.11%	9.511×10^{-7}	0.02%	3.8×10^6	37.26%
1000	0.1%	6.2×10^{-5}	0.01%	0.001	0.05%	9.456×10^{-7}	0.018%	9×10^6	26.56%

Table 4.4: Percentage of taken flows given a portion of sampled packets for trace UPC-I

N	%packets	*Selective Sampling*			
		z	c	n	%flows
10	10%	6	1.0	2	87.9396%
50	2%	10	0.2	150	18.575%
100	1%	10	0.1	400	9.33%
500	0.2%	1	0.06	100	3.801%
1000	0.1%	1	0.03	250	1.902%

Table 4.5: Percentage of taken packets given a portion of sampled flows for trace UPC-I

N	%flows	*Selective Sampling*			
		z	c	n	%packets
10	10%	1	0.08	2	1.294%
50	2%	1	0.03	70	0.120%
100	1%	1	0.001	11	0.191%
500	0.2%	1	0.001	73	0.0305%
1000	0.1%	1	0.001	276	0.01061%

Chapter 5

Analyzing the Impact of Sampling on Portscan Detection

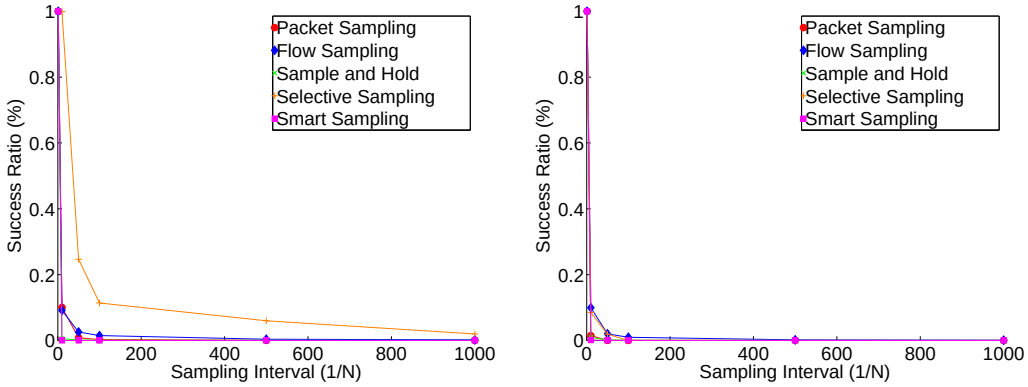
In this section, we study the impact of *Packet Sampling*, *Flow Sampling*, *Sample and Hold*, *Smart Sampling* and *Selective Sampling* sampling techniques on TRW and TAPS portscan detection algorithms. We used the following performance metrics:

$$success\_ratio = \frac{true\_scanners}{total\_scanners} \qquad \text{and} \qquad false\_positive\_ratio = \frac{false\_scanners}{total\_scanners},$$

where *total_scanners* accounts for our ground truth of scanners (scanners detected by TRW/TAPS without sampling, which are not necessarily real scanners). While *true_scanners* stands for the scanners detected under sampling that also belong to the ground truth, *false_scanners* refers to those detected scanners that fall out of that set. Note that our metrics differ from the classical definitions of success and false positive ratios in that we do not check whether the detected scanners by TRW and TAPS (without sampling) are real scanners or not. This choice lies in the fact that we are interested in evaluating the degradation of the portscan detection algorithms in the presence of sampling rather than in their actual detection accuracy.

5.1 Impact of sampling on TRW

Table 5.1 shows the total number of hosts classified as scanners by TRW for every trace. Tables 5.2, 5.3, 5.4, 5.5 and 5.6 show TRW per-trace performance under *PS*, *FS*, *SH*, *SMS* and *SES* respectively. The results for each trace are splitted into those obtained using the same fraction of packets (left side) and flows (right side). The last column of each of these two groups (TS) indicates the total num-



(a) Using equal fraction of packets (b) Using equal fraction of flows

Fig. 5.1: Impact of sampling on TRW's Success Ratio (average among all the traces)

Table 5.1: Total scanners detected by TRW without sampling

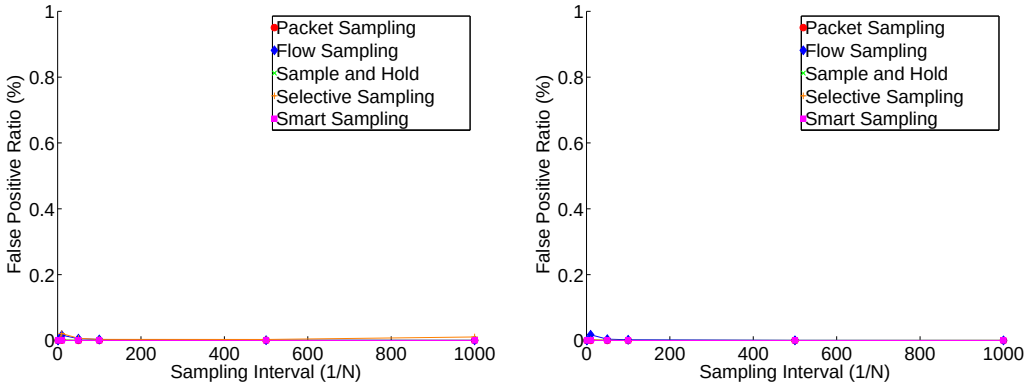
Trace	Total Scanners
UPC-I	14221
UPC-II	8476
UPC-III	10666
UPC-IV	13025
UPC-V	14540

ber of hosts flagged as scanners using that particular sampling method, sampling parameters, common metric (%packets or %flows) and NetFlow data trace.

5.1.1 Impact of sampling on TRW Success Ratio

As we can observe in Figure 5.1, success ratio degrades dramatically for increasing sampling rates regardless of the common metric (portion of packets or flows) and the sampling technique used. When the sampling rate is low, TRW still detects few scanners but when it goes up, the success ratio reaches zero quickly.

Looking at Figure 5.1(a), we found that sampling methods targeting large flows (*SMS* and *SH*) are not suitable for portscan detection since this kind of attack is made using small flows. They both detected almost nothing for all the sampling intervals. Concerning the other sampling techniques we were able to observe two differentiated behaviours. In the first place, *PS* and *FS* showed similar performance degradations when sampling 10% and 2% of the packets respectively ($N = 10$ and $N = 50$). For larger values of N, *FS* slightly outperforms *PS*. On the other hand, *Selective Sampling* shows a huge improvement in the success ratio in comparison to the rest of sampling techniques, degrading smoothly with sampling



(a) Using equal fraction of packets (b) Using equal fraction of flows

Fig. 5.2: Impact of sampling on TRW's False Positive Ratio (average among all the traces)

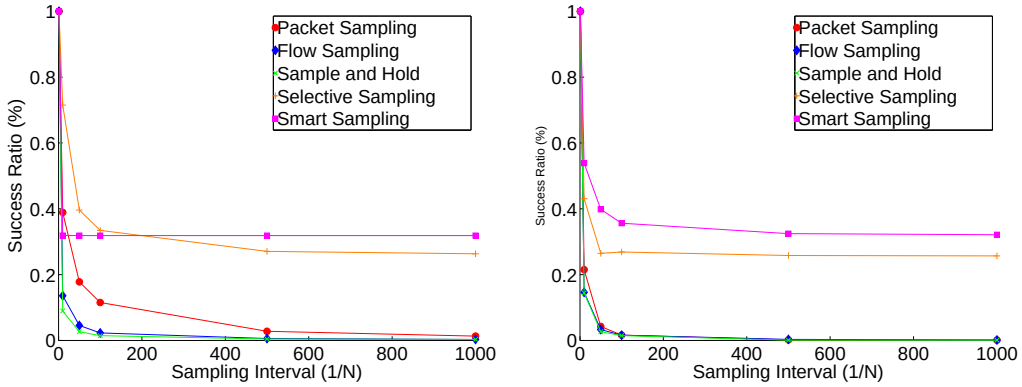
rate. This is due to the fact that it targets small flows, thus sampling exactly the essential flows for portscan detection.

Figure 5.1(b) shows again the severe impact of sampling on the performance of TRW. In contrast to the previous case (same fraction of packets), when using the same fraction of flows, the behaviour for all the sampling techniques is almost indistinguishable. The ones that showed to be more sampling-resilient were *Flow Sampling*, and *SES* while $N \leq 50$, i.e., while sampling 2% of the flows at most. The other sampling techniques detected almost nothing.

5.1.2 Impact of sampling on TRW False Positive Ratio

Switching to the false positives ratio, Figure 5.2 show that it is relatively low ($\approx 2\%$ in the worst case).

Figure 5.2(a) (same fraction of packets) shows that the false positives exhibited by *PS*, *SH* and *SMS* are almost negligible. There are two peaks: $\approx 1\%$ for *FS* and almost 2% for *SES*. *Selective Sampling* false positives could be explained because of the following phenomenon. Since it focuses in small flows and it looks down on large ones, multi-packet flows from successful connections are dropped with higher probability, thus making TRW receive mainly failed connections. Even if a host had legitimate connections, it is very likely for TRW to miss them as if they had never existed, thus contributing to potentially flag as a scanner a host with some maybe unintentional failed connections. Looking at the other sampling intervals we can see how *SES* false positives decrease rapidly while $N \leq 100$. For higher values of N it showed different behaviours among traces. While in some cases,



(a) Using equal fraction of packets (b) Using equal fraction of flows

Fig. 5.3: Impact of sampling on TAPS's Success Ratio (average among all the traces)

it went down, there are traces where fpr increased (see table 5.6). In average it seemed to be constant within this range. For sampling rates lower than 0.2% of the packets, it exhibited a tendency change clearly raising up again. This differentiated behaviour suggests that the before explained phenomenon is more visible since the more aggressive is the sampling rate applied, the lower is the probability of taking non-small flows, thus emphasizing the problem for TRW.

Concerning to the same portion of traffic in terms of flows (Figure 5.2(b)), *FS* shows again a little peak a little bit lower than 2% while the other techniques show really small values.

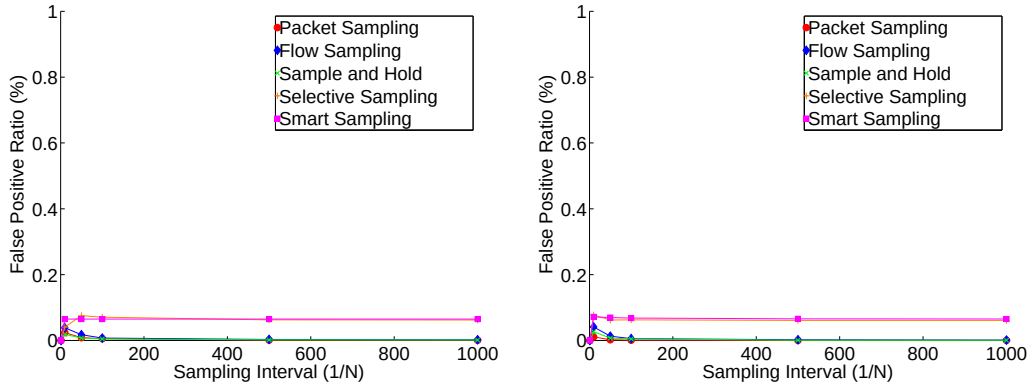
5.1.3 Studying TRW performance degradation

Traffic Distribution

In order to understand why TRW had this poor performance, we started observing the behaviour of scanners and legitimate hosts by monitoring how their connections ended (either being successful attempts or failures).

There is a non-negligible part of benign hosts that have a significant number of consecutive failed flows. A priori, it seems highly unlikely that benign hosts generate so many consecutive failed connections. This means that, previously to that burst of failed flows, those hosts made a number of non-failed connections large enough to make TRW classify them as non-scanners. In order to see what happened in these particular cases, we proceeded looking flow by flow for those suspicious hosts classified as legitimate.

Indeed, the before mentioned hosts, previously to the burst of failed flows,



(a) Using equal fraction of packets (b) Using equal fraction of flows

Fig. 5.4: Impact of sampling on TAPS's False Positive Ratio (average among all the traces)

had a sufficient number of successful connections that made TRW classify them as benign. Things changed when we checked those source IPs with sampling. It turns out that, in some cases, sampling helps uncovering those malicious hosts. That happens because some of that benign flows that made TRW to take a wrong decision, now, are gone because of sampling. This phenomenon explains part of the false positives. Those firstly erroneously classified IPs that belong to the ground truth are then correctly classified as scanners but they do not match with the ground truth of scanners, thus contributing to increase the false positive ratio.

The most extreme case we found was a source IP X that had 4487 consecutive failed connections in the unsampled case and was classified as a benign host. All the sampling methods finally classified it as a scanner but it changed how soon they did that. That IP is classified as non-scanner even sampling 10%, 2% and 1% of packets for *Packet Sampling*. Finally, TRW classifies it correctly when sampling 0.2% and 0.1% of the packets (using *PS*). Looking for other cases we found a source IP Y that with 60 consecutive failed connection was a another clearly wrong classified host. In contrast to the host X, when we applied sampling, it was only correctly classified as a scanner for one sampling method: *Selective Sampling*. Other IPs with a lower but still more than suspicious number of consecutive failed connections showed the same behaviour of host Y. Except for one case in which *PS* classified a host correctly just for a particular sampling rate, all the other sampling methods except for *SES* missed these scanners for all sampling rates. Therefore, it seems that if the consecutive number of failed of connections is not large enough, almost all sampling methods clearly do not flag some real scanners (except for *Selective Sampling*).

Table 5.2: TRW performance on each trace using *Packet Sampling*. TS stands for total scanners detected

Trace	% pkts	SR	FPR	TS	% flows	SR	FPR	TS
UPC-I	10%	8.97%	0.008%	1287	10%	1.2%	0.02%	173
	2%	0.8%	0.01%	116	2%	0.02%	0.01%	4
	1%	0.23%	0.01%	34	1%	0%	0.01%	1
	0.2%	0.01%	0.01%	2	0.2%	0%	0%	0
	0.1%	0.01%	0.01%	2	0.1%	0%	0%	0
UPC-II	10%	11.4%	0.14%	978	10%	1.77%	0.02%	152
	2%	1.09%	0.01%	93	2%	0.07%	0%	6
	1%	0.35%	0%	30	1%	0%	0.02%	2
	0.2%	0.08%	0%	7	0.2%	0%	0%	0
	0.1%	0.01%	0%	1	0.1%	0%	0%	0
UPC-III	10%	9.34%	0.09%	1006	10%	1.37%	0.01%	147
	2%	0.73%	0%	78	2%	0.03%	0%	3
	1%	0.23%	0%	25	1%	0%	0%	0
	0.2%	0.03%	0%	3	0.2%	0%	0%	0
	0.1%	0%	0%	0	0.1%	0%	0%	0
UPC-IV	10%	12.4%	0.08%	1625	10%	1.73%	0.02%	228
	2%	0.74%	0.02%	99	2%	0.03%	0%	4
	1%	0.25%	0.01%	33	1%	0%	0%	0
	0.2%	0.02%	0%	2	0.2%	0%	0%	0
	0.1%	0%	0%	0	0.1%	0%	0%	0
UPC-V	10%	9.95%	0.1%	1462	10%	1.23%	0.03%	183
	2%	0.66%	0.01%	98	2%	0.03%	0%	5
	1%	0.23%	0%	33	1%	0%	0%	0
	0.2%	0.01%	0%	2	0.2%	0%	0%	0
	0.1%	0%	0%	0	0.1%	0%	0%	0

The reason why that happens is because *SES* focuses on small flows. For non-small flows, the more packets they have, the higher is the chance of discarding them. *Sample an Hold* and *Smart Sampling* target large flows, so the more aggressive is the sampling, the worse (single SYN-packet flows are dropped). Regarding *Packet Sampling* and *Flow Sampling* we observed that they are not good enough either.

θ_0 and θ_1 variations

We wanted to see if the hypothesis default values were still valid under sampling. We first ran TRW using the original probability values (0.8 and 0.2) under increasing sampling rates. Using the unsampled case as our ground truth of scanners, we re-computed θ_0 and θ_1 for each sampling rate ($srate$).

For instance, in order to compute the new hypothesis values under 10% of sampled packets, we first ran TRW with $\theta_0 = 0.8$ and $\theta_1 = 0.2$ and $srate = 0.1$. After that, we compared the sampled case output against the unsampled case output (ground truth). For each true scanner detected we accumulated both the number of failed and successful connections. The same was done for the

Table 5.3: TRW performance on each trace using *Flow Sampling*. TS stands for total scanners detected

Trace	% pkts	SR	FPR	TS	% flows	SR	FPR	TS
UPC-I	10%	8.92%	1.04%	1287	10%	9.75%	1.11%	1545
	2%	2.58%	0.34%	116	2%	1.97%	0.25%	315
	1%	1.45%	0.18%	34	1%	1%	0.11%	158
	0.2%	0.43%	0.04%	2	0.2%	0.22%	0.02%	34
	0.1%	0.19%	0.01%	2	0.1%	0.10%	0.01%	15
UPC-II	10%	9.59%	1.81%	966	10%	10.46%	1.9%	1048
	2%	2.71%	0.47%	270	2%	2.1%	0.34%	0.07
	1%	1.55%	0.28%	115	1%	1.06%	0.19%	106
	0.2%	0.37%	0.07%	37	0.2%	0.2%	0.04%	20
	0.1%	0.19%	0.04%	19	0.1%	0.11%	0%	9
UPC-III	10%	9.03%	1.88%	1163	10%	9.7%	1.95%	1243
	2%	2.74%	0.66%	362	2%	2.21%	0.49%	288
	1%	1.62%	0.38%	213	1%	1.07%	0.25%	141
	0.2%	0.39%	0.12%	55	0.2%	0.22%	0.06%	29
	0.1%	0.19%	0.06%	26	0.1%	0.11%	0.02%	14
UPC-IV	10%	8.81%	1.52%	1345	10%	9.75%	1.6%	1478
	2%	2.37%	0.51%	376	2%	1.83%	0.39%	289
	1%	1.37%	0.29%	217	1%	0.78%	0.14%	119
	0.2%	0.31%	0.06%	49	0.2%	0.15%	0.02%	22
	0.1%	0.12%	0.02%	19	0.1%	0.09%	0.01%	13
UPC-V	10%	9.06%	1.53%	1541	10%	9.88%	1.71%	1685
	2%	2.48%	0.52%	436	2%	1.86%	0.38%	326
	1%	1.37%	0.3%	242	1%	0.84%	0.18%	148
	0.2%	0.27%	0.08%	50	0.2%	0.17%	0.03%	29
	0.1%	0.16%	0.03%	27	0.1%	0.06%	0.01%	11

legitimate hosts. Subsequently, we computed the average for each of the four sets: probability of failed connections from legitimate hosts (θ_1), probability of successful connections from legitimate hosts (1-θ_1) and the same for the scanners ($1-\theta_0$ and θ_0 probabilities respectively).

As we can observe in table 5.7, both probabilities went up with increasing sampling rates. This resulted in differentiated behaviours of TRW according to the sampling sampling rate being applied. For instance, in the 10% sampling case, it means that TRW will take faster decisions since the update value for failed connections is now 6.9, almost twice the original value (4). The algorithm will take a decision sooner. The same phenomenon happens for non-scanner hosts, where the decrease is quicker (0.18) than the original (0.25).

- Failed connection:
  - $\bigwedge(Y) \equiv \frac{Pr[Y=1|H_1]}{Pr[Y=1|H_0]}$
  - Without sampling: $\bigwedge(Y) = \frac{0.8}{0.2} = 4$
  - 10% sampling: $\bigwedge(Y) = \frac{1-0.1572}{1-0.8778} = \frac{0.8428}{0.122} \approx 6.9$
- Successful connection:

Table 5.4: TRW performance on each trace using *Sample and Hold*. TS stands for total scanners detected

Trace	% pkts	SR	FPR	TS	% flows	SR	FPR	TS
UPC-I	10%	0.57%	0.04%	87	10%	0.92%	0.1%	145
	2%	0.15%	0.01%	24	2%	0.15%	0.01%	24
	1%	0.08%	0%	12	1%	0.08%	0%	12
	0.2%	0.03%	0%	4	0.2%	0.01%	0%	1
	0.1%	0.01%	0%	2	0.1%	0%	0%	0
UPC-II	10%	0.59%	0.13%	61	10%	1.01%	0.15%	99
	2%	0.17%	0.02%	16	2%	0.17%	0.02%	16
	1%	0.09%	0%	8	1%	0.09%	0%	8
	0.2%	0.05%	0%	4	0.2%	0%	0%	0
	0.1%	0.04%	0%	3	0.1%	0%	0%	0
UPC-III	10%	0.65%	0.17%	87	10%	0.99%	0.21%	128
	2%	0.14%	0.04%	18	2%	0.14%	0.04%	18
	1%	0.08%	0.02%	10	1%	0.08%	0.02%	10
	0.2%	0.05%	0.01%	6	0.2%	0%	0.01%	1
	0.1%	0.05%	0.01%	6	0.1%	0%	0.01%	1
UPC-IV	10%	0.46%	0.06%	68	10%	0.77%	0.1%	113
	2%	0.1%	0.02%	15	2%	0.1%	0.02%	15
	1%	0.08%	0.01%	15	1%	0.08%	0.01%	15
	0.2%	0.02%	0%	3	0.2%	0.02%	0%	2
	0.1%	0.02%	0%	2	0.1%	0.02%	0%	2
UPC-V	10%	0.44%	0.1%	79	10%	0.81%	0.15%	140
	2%	0.11%	0.02%	19	2%	0.11%	0.02%	19
	1%	0.04%	0.01%	8	1%	0.04%	0.01%	8
	0.2%	0.02%	0%	3	0.2%	0.01%	0%	2
	0.1%	0.01%	0%	2	0.1%	0.01%	0%	2

- $\bigwedge(Y) \equiv \frac{Pr[Y=0|H_1]}{Pr[Y=0|H_0]}$
- Without sampling: $\bigwedge(Y) = \frac{0.2}{0.8} = 0.25$
- 10% sampling: $\bigwedge(Y) = \frac{0.1572}{0.8778} \approx 0.18$

This subsection shows that TRW will be always incapable of working properly under sampling without any modification regarding its hypothesis probabilities (at least). As explained before, probabilities change dramatically with varying sampling rates thus making default TRW hypothesis values (0.8 and 0.2) almost useless for portscan detection under sampling.

5.1.4 Trying to improve TRW's performance under sampling

In order to observe how TRW would work under sampling, we first ran our experiments with the same original configuration parameters values of the unsampled case. Therefore, for all sampling rates, we used the recommended hypothesis values ($\theta_0 = 0.8$ and $\theta_1 = 0.2$) together with the default false positives and the detection probability values (0.01 and 0.99 respectively).

Table 5.5: TRW performance on each trace using *Smart Sampling*. TS stands for total scanners detected

Trace	% pkts	SR	FPR	TS	% flows	SR	FPR	TS
UPC-I	10%	0%	0%	0	10%	0.01%	0.5%	15
	2%	0%	0%	0	2%	0%	0%	0
	1%	0%	0%	0	1%	0%	0.01%	0
	0.2%	0%	0%	0	0.2%	0%	0%	0
	0.1%	0%	0%	0	0.1%	0%	0%	0
UPC-II	10%	0%	0%	0	10%	0.13%	0%	11
	2%	0%	0%	0	2%	0.01%	0%	1
	1%	0%	0%	0	1%	0%	0%	0
	0.2%	0%	0%	0	0.2%	0%	0%	0
	0.1%	0%	0%	0	0.1%	0%	0%	0
UPC-III	10%	0%	0%	0	10%	0.04%	0%	4
	2%	0%	0%	0	2%	0%	0%	0
	1%	0%	0%	0	1%	0%	0%	0
	0.2%	0%	0%	0	0.2%	0%	0%	0
	0.1%	0%	0%	0	0.1%	0%	0%	0
UPC-IV	10%	0%	0%	0	10%	0.08%	0%	10
	2%	0%	0%	0	2%	0%	0%	0
	1%	0%	0%	0	1%	0%	0.01%	0
	0.2%	0%	0%	0	0.2%	0%	0%	0
	0.1%	0%	0%	0	0.1%	0%	0%	0
UPC-V	10%	0%	0%	0	10%	0.06%	0%	9
	2%	0%	0%	0	2%	0%	0%	0
	1%	0%	0%	0	1%	0%	0%	0
	0.2%	0%	0%	0	0.2%	0%	0%	0
	0.1%	0%	0%	0	0.1%	0%	0%	0

TRW performance degraded dramatically with increasing sampling rates. Actually, even under the lowest sampling rate that we used (10% of flows or packets), the algorithm was almost blind (very few scanners detected). In order to attempt to obtain a better performance we started monitoring how its behaviour degradation was related to the applied sampling rate.

Modifying TRW's update mechanism

The only techniques that allow to know the percentage of traffic that you will sample before analyzing a particular trace are *Packet Sampling* and *Flow Sampling*. In the first case it permits to know the fraction of packets that will be keeped, while in the second one, it indicates the portion of flows that will be taken. This fact is important because the update proposed in this section is directly related to the fraction of traffic keeped. Because of that, this section only applies to these two methods, because with the others is impossible to know the traffic that you will keep, thus making impossible to correct TRW's update mechanism accordingly.

When there is no sampling, TRW needs a certain number of failed connections to flag a scanner. With sampling, it is really more difficult for TRW to catch

Table 5.6: TRW performance on each trace using *Selective Sampling*. TS stands for total scanners detected

Trace	% pkts	SR	FPR	TS	% flows	SR	FPR	TS
UPC-I	10%	99.94%	1.74%	14460	10%	7.07%	0.05%	1013
	2%	23.37%	0.27%	3363	2%	1.61%	0.04%	235
	1%	9.30%	0.10%	1337	1%	0.01%	0%	1
	0.2%	4.82%	0.17%	710	0.2%	0%	0%	0
	0.1%	1.55%	0.05%	228	0.1%	0%	0.01%	1
UPC-II	10%	99.92%	2.29%	8663	10%	9.79%	0.21%	848
	2%	27.24%	0.85%	2381	2%	2.23%	0.07%	195
	1%	13.37%	0.45%	1171	1%	0.06%	0%	11
	0.2%	7.39%	0.33%	654	0.2%	0.05%	0%	4
	0.1%	2.67%	1.37%	342	0.1%	0.02%	0%	2
UPC-III	10%	99.85%	1.64%	10825	10%	8.45%	0.23%	926
	2%	24.01%	0.61%	2626	2%	2.04%	0.02%	220
	1%	11.36%	0.29%	1243	1%	0.02%	0%	2
	0.2%	6.07%	0.33%	682	0.2%	0.01%	0%	1
	0.1%	2.04%	1.09%	334	0.1%	0.01%	0%	1
UPC-IV	10%	99.96%	2.39%	13331	10%	10.73%	0.21%	1424
	2%	28.48%	0.68%	3799	2%	2.23%	0.07%	299
	1%	14%	0.32%	1866	1%	0%	0%	0
	0.2%	7.12%	0.03%	967	0.2%	0%	0%	0
	0.1%	2.29%	1.57%	503	0.1%	0.01%	0%	1
UPC-V	10%	99.88%	2.68%	14911	10%	8.47%	0.28%	1272
	2%	24.55%	0.97%	3710	2%	1.86%	0.07%	281
	1%	11.62%	0.47%	1758	1%	0.01%	0%	1
	0.2%	5.85%	0.35%	902	0.2%	0.01%	0%	1
	0.1%	1.69%	1.11%	407	0.1%	0.01%	0.01%	1

the needed number of failed connections because of the discarded packets. Due to that fact, TRW is working under unfair conditions, because it needs far more failed connections than in the unsampled case. Somehow, TRW should account for those thrown packets. For instance, sampling 10% of the flows means that if we saw one flow from a certain source IP it is like we had 10 original flows from that host but due to sampling, we are only seeing one of them. In order to use this observation in TRW, we modified its update mechanism in such a way that sampling is taking into account:

- Original update: $\bigwedge(Y) = \bigwedge(Y) \times update\_value$
- With sampling: $\bigwedge(Y) = \bigwedge(Y) \times update\_value^{\frac{1}{sampling\_rate}}$

Re-taking the 10% sampling example, this update would result in the same final ratio as executing original TRW for 10 times. Considering default TRW values, if a single SYN-packet flow is found the update would be: $4^{\frac{1}{0.1}} = 4^{10}$. Otherwise, it would result in $0.25^{\frac{1}{0.1}} = 0.25^{10}$. In both cases the final value of the update is far higher/lower than the default upper/lower threshold respectively. Hence, both updates would imply that TRW would classify a particular host using just a single

Table 5.7: θ_0 and θ_1 variation due to sampling rate applied using UPC-I trace

% of packets	θ_0	θ_1
10%	87.78%	15.72%
2%	92.13%	28.24%
1%	93.88%	25.61%
0.2%	96.85%	29.17%
0.1%	97.49%	38.89%

observation. If we increase the sampling rate, the exponent would be even higher, so it would result in a single step classification as well.

This phenomenon makes TRW almost useless because the single step decision is totally contradictory with the sequential hypothesis testing method used by the portscan detection algorithm. The idea is that TRW tracks the connections from a host, but with this update, it turns out that there is no such tracking because a single flow is enough to classify any host.

The problem with the proposed update is that we could not be always sure that the failed connection found in the sampled traffic was really an original failed connection. Therefore, we had to determine when this failed connection was indeed original. In the case of *Flow Sampling*, since it either takes all the flow or it does not even take a single packet of it, a failed connection found in the sampled traffic implies that it is an original flow for sure. When switching to *Packet Sampling* this affirmation is not true. A single SYN-packet flow (failed connection) could correspond to a non-original failed flow because *PS* does not keep the flow distribution. Hence, we can have a failed connection in the sampled input that was really a multi-packet flow coming from a legitimate client.

For *FS*, the update mechanism proposed before is still correct, but for *PS*, some more adjustments must be made according to the probability of a single SYN-packet flow of being original (see Table 5.8). We adjust the before proposed update mechanism with these probabilities (*prob_original_sspf*) in such a way that we modify the update value according to both the sampling rate and the probability of a failed connection of being true.

- With sampling: $\bigwedge(Y) = \bigwedge(Y) \times update\_value^{\frac{prob\_original\_sspf}{srate}}$

The new variable added to the formula allows to further regulate the exponent of the update according to the percentage of real single SYN-packet flows found. For instance, instead of proceeding as in the example before when detecting a failed connection ($4^{\frac{1}{0.1}} = 4^{10}$), the update would be as follows:

$$4^{\frac{0.09}{0.1}} = 4^{0.9} = 3.48$$

In this case, in contrast to what happened with *FS*, the update slows down TRW rather than speeding it up.

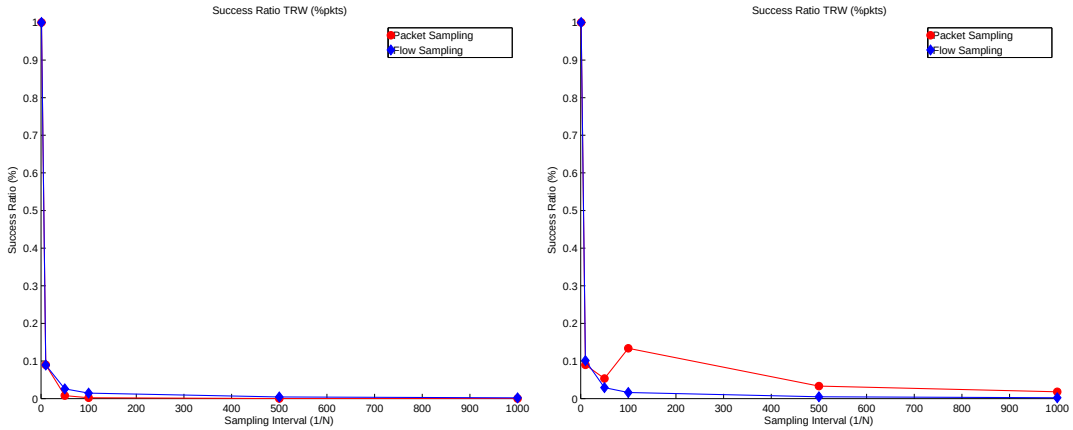


Fig. 5.5: Impact of sampling on TRW's Success Ratio using equal fraction of packets on trace UPC-I running TRW without any modification (left) and modifying TRW as explained in this section (rigth)

Table 5.8: Probability of finding an original single syn-packet flow (sspf) using *Packet Sampling* and a particular sampling rate for UPC-I trace

% of packets	prob_original_sspf
10%	9%
2%	6%
1%	5%
0.2%	5%
0.1%	5%

On Figures 5.5, 5.6, 5.7 and 5.8 we can observe the performance degradation of TRW using UPC-I trace. The original TRW performance is plotted on the left side and the modified one on the right. The success ratios (Figures 5.5 and 5.6) are improved using both metrics (equal fraction of packets and flows) but in a different measure. The most significant improvement happens in the case of *PS* using the same fraction of packets. For the other combinations, even that it is not so easy to observe, the performance is slightly better. In exchange for this performance improvement, it arises a huge increase of false positives. Using the same fraction of packets (Figure 5.7), *FS* maximum false positive rate changes from $\approx 1\%$ for the original case to almost 30% for the updated TRW. *PS* reached 25% while with the original TRW it showed a value below 0.1%. The same phenomenon happened using the same fraction of flows (look at Figure 5.8).

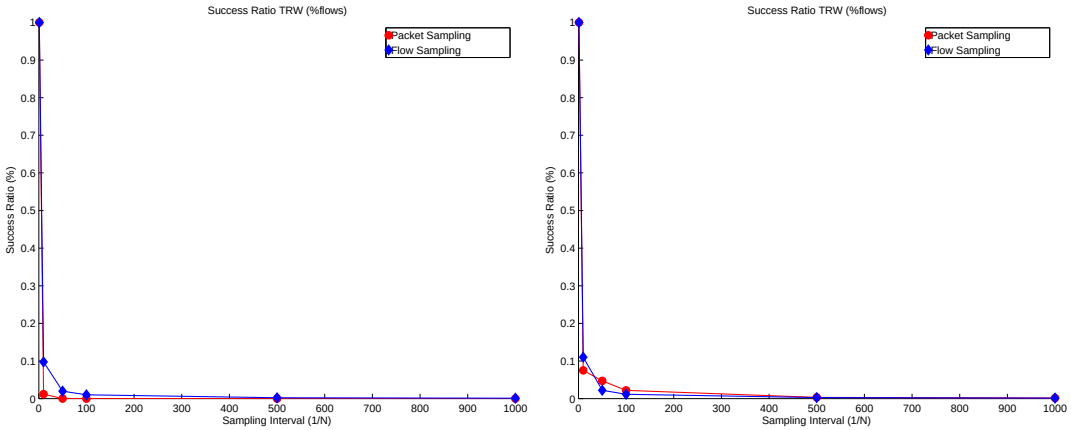


Fig. 5.6: Impact of sampling on TRW's Success Ratio using equal fraction of flows on trace UPC-I running TRW without any modification (left) and modifying TRW as explained in this section (rigth)

Table 5.9: Total scanners detected by TAPS without sampling

Trace	Total Scanners
UPC-I	10387
UPC-II	6284
UPC-III	8712
UPC-IV	13762
UPC-V	9649

5.2 Impact of sampling on TAPS

Table 5.9 shows the total number of hosts classified as scanners by TAPS for every trace. Tables 5.10, 5.11, 5.12, 5.13 and 5.14 show TAPS per-trace performance under *PS*, *FS*, *SH*, *SMS* and *SES* respectively. The results for each trace are splitted into those obtained using the same fraction of packets (left side) and flows (right side). The last column of each of these two groups (TS) indicates the total number of hosts flagged as scanners using that particular sampling method, sampling parameters, common metric (%packets or %flows) and NetFlow data trace.

5.2.1 Impact of sampling on TAPS Success Ratio

Figure 5.3 shows how TAPS performed for increasing sampling rates regarding the success ratio.

As we can observe in Figure 5.3(a), when using the same fraction of packets,

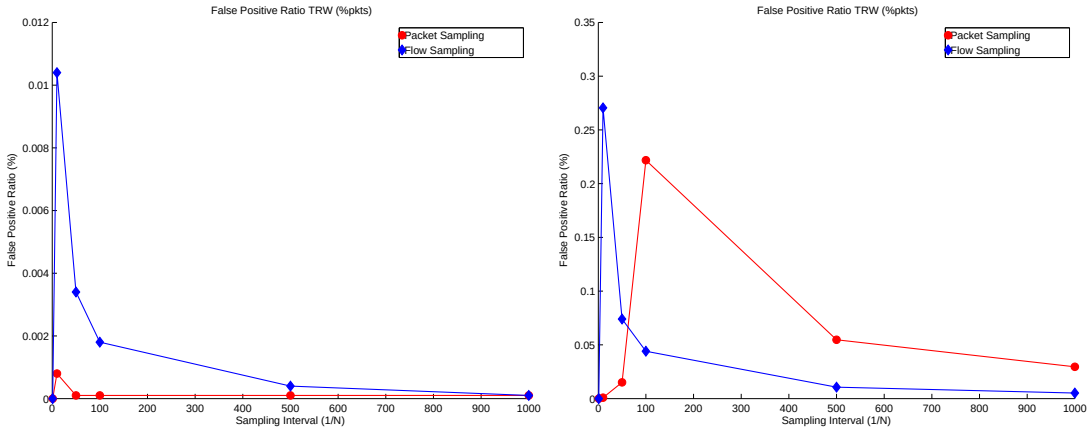


Fig. 5.7: Impact of sampling on TRW's False Positive Ratio using equal fraction of packets on trace UPC-I running TRW without any modification (left) and modifying TRW as explained in this section (rigth)

there are different behaviours among the sampling methods. *SH* and *Flow Sampling* presented the highest impact detecting almost no scanners when sampling more than 1% of the packets. Looking at higher sampling rates they did not perform better either: with 10% of packets *FS* hardly achieved a 15% of success ratio while *Sample and Hold* did not even got 10%. *Packet Sampling* degraded smoothly with sampling getting almost 40% of success for 10% of the packets and a $sr \geq 10\%$ for $10 < N \leq 100$. It even detected some scanners for $N \approx 1000$. Regarding *SMS* and *SES*, the performance exhibited was extraordinary. As we can see on the picture, they both degraded until a particular sampling interval and from that point they maintained a constant success ratio until the end ($sr \approx 35\%$ and 30% respectively). There is a key point from where selected flows do not affect scanners at all.

As depicted in Figure 5.3(b), when using the same fraction of flows *SH*, *PS* and *FS* exhibited nearly indistinguishable behaviours. *Packet Sampling* turned out to be slightly better for lower sampling rates. Even for the former method, the achieved success ratios were totally devastated for these three sampling techniques. Concerning to the remaining two (*SMS* and *SES*), it occurs the same as in the case of using the same fraction of packets. They reach a sampling interval point from where the sampled traffic has nothing to do with scanners, thus keeping the same ratio of detection.

5.2.2 Impact of sampling on TAPS False Positive Ratio

Figure 5.4 presents TAPS false positives ratios under different sampling rates.

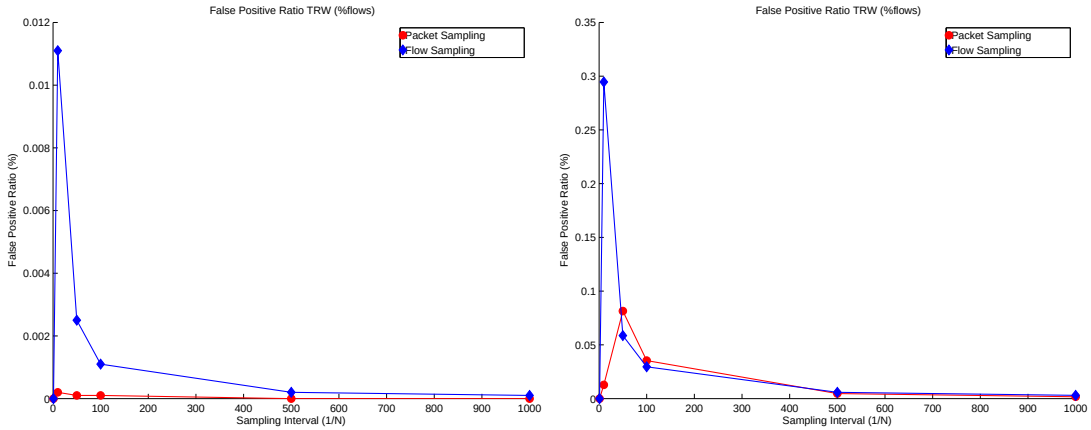


Fig. 5.8: Impact of sampling on TRW's False Positive Ratio using equal fraction of flows on trace UPC-I running TRW without any modification (left) and modifying TRW as explained in this section (rigth)

Using equal fraction of packets (Figure 5.4(a)), we observed clear peaks for $N = 10$ for *SH PS* i *FS* that almost reached 4% in the worst case. For higher values of N the false positives went down according to the sampling rate. *SMS* and *SES* exhibited constant and very similar false positive ratios ($\approx 7\%$) regardless of the sampling rate.

Switching to the same portion of flows (Figure 5.4(a)), the presented behaviour was very similar to the same fraction of packets case. *SMS* and *SES* exhibited an almost equal number of false positives ($< 10\%$) all the time and the other sampling techniques got almost negligible values except for $n = 10$, in which case they reached $\approx 5\%$.

5.2.3 Studying TAPS performance degradation

Even for the unsampled case, the two parameters used by TAPS (the time bin t used to trigger the per source IP destination patterns checking and the threshold k used to determine suspicious behaviour) are crucial. With sampling, these two variables turned out to be even more important. They should be adjusted to further adapt TAPS to a particular sampling method and rate. In order to to that, we will have to study how the destination access patterns changed with sampling. So far, we have been able to see that it changes significantly but in a different way depending on the sampling method used. We will also have to investigate how important is t under sampling and if it depends on the concrete sampling mechanism. All this is a part of our on-going work.

Table 5.10: TAPS performance on each trace using *Packet Sampling*. TS stands for total scanners detected

Trace	% pkts	SR	FPR	TS	% flows	SR	FPR	TS
UPC-I	10%	49.3%	2.11%	5340	10%	28.62%	1.01%	3078
	2%	24.52%	0.89%	2639	2%	6.69%	0.18%	714
	1%	16.81%	0.59%	1807	1%	2.31%	0.08%	248
	0.2%	4.21%	0.13%	450	0.2%	1.6%	0.01%	18
	0.1%	1.83%	0.06%	196	0.1%	0.09%	0%	9
UPC-II	10%	30.52%	2.45%	2072	10%	14.97%	0.99%	1003
	2%	12.06%	0.67%	800	2%	2.85%	0.14%	188
	1%	7.77%	0.32%	508	1%	1.18%	0.03%	76
	0.2%	2.08%	0.08%	136	0.2%	0.11%	0.02%	8
	0.1%	1.02%	0%	64	0.1%	0.05%	0%	3
UPC-III	10%	25.59%	2.2%	2421	10%	12.65%	0.87%	1178
	2%	10.04%	0.61%	928	2%	2.39%	0.16%	222
	1%	6.18%	0.34%	568	1%	0.93%	0.06%	86
	0.2%	1.53%	0.09%	141	0.2%	0.07%	0%	6
	0.1%	0.77%	0.05%	71	0.1%	0.05%	0.01%	5
UPC-IV	10%	48.18%	2.63%	6993	10%	28.73%	1.45%	4153
	2%	23.8%	1.16%	3435	2%	5.29%	0.28%	767
	1%	15.35%	0.83%	2227	1%	2%	0.17%	298
	0.2%	3.45%	0.21%	504	0.2%	0.28%	0.01%	39
	0.1%	1.66%	0.1%	242	0.1%	0.15%	0%	21
UPC-V	10%	40.9%	2.44%	4181	10%	22.55%	1.11%	2283
	2%	18.57%	0.91%	1880	2%	3.99%	0.19%	403
	1%	11.49%	0.54%	1161	1%	1.62%	0.06%	162
	0.2%	2.6%	0.08%	2	0.2%	0.11%	0.01%	12
	0.1%	1.32%	0.04%	0	0.1%	0.03%	0%	3

5.3 TRW vs. TAPS

It is obvious that both TRW and TAPS performance were seriously affected. However, the damage suffered by TAPS was not as severe as in the TRW case, which obtained really poor results with sampling. There are many reasons why this happened. As already noticed by previous studies, under sampling, we were able to detect many more scanners using TAPS than TRW. Furthermore, TRW showed to be much less resilient to sampling, while TAPS detected scanners even sampling 0.1% of the traffic ($N = 1000$). TAPS does not depend on any specific packet feature like TRW (which looks for single SYN-packet flows), thus being significantly less sensitive to the particular packet being discarded. This can be explained partially due to the fact that TRW only works with TCP scanners and TAPS is connectionless-oriented (it works for both UDP and TCP).

Looking at the success ratio (sr), it is clear that TAPS is far better than TRW. The unique acceptable sr that TRW showed was using the same fraction of packets and *Selective Sampling* (Figure 5.1(a)). For all the other cases, the results were really disappointing, hardly achieving $sr = 10\%$ for PS and FS with 10% of sampled packets and going almost blind for more aggressive sampling rates. The results

using equal percentage of sampled flows were even slightly worse (Figure 5.1(b)). The success ratios showed by TAPS were good enough for all sampling methods. The worst performance cases (showed by *FS* and *SH*) detected some scanners for $N = 10$, 50 and 100 but for $N > 100$ they detected nothing. *Packet Sampling* degrades proportionally to sampling rate showing a really good performance even for the maximum sampling rate ($N = 1000$). While *SES* showed to be the best global (for both TRW and TAPS) sampling technique, *SMS* presented totally opposite results. With TRW it is absolutely useless because it drops the flows that TRW needs to do the detections, but in the case of TAPS it seems to work remarkably well almost regardless of the sampling rate.

Concerning the false positive ratio (fpr), TAPS obtained significantly worse values than TRW. While the former hardly reached 2% of false positive in the worst case (Figure 5.2(a)), TAPS almost obtained $fpr = 10\%$ using the same fraction of flows (Figure 5.4(b)) and $fpr = 8\%$ with equal portion of packets (Figure 5.4(a)).

Taking all this into account, we can conclude that TAPS is better under sampling as already noticed by previous works. Furthermore, in contrast to what former studies concluded, when using the same fraction of packets as the common metric to compare the different sampling methods under TAPS, we obtained significantly better results using *Packet Sampling* than with *Flow Sampling*.

Table 5.11: TAPS performance on each trace using *Flow Sampling*. TS stands for total scanners detected

Trace	% pkts	SR	FPR	TS	% flows	SR	FPR	TS
UPC-I	10%	11.81%	2.39%	1475	10%	12.67%	2.52%	1578
	2%	3.85%	0.88%	491	2%	3.14%	0.69%	398
	1%	2.3%	0.52%	293	1%	1.71%	0.39%	218
	0.2%	0.77%	0.27%	108	0.2%	0.38%	0.18%	58
	0.1%	0.31%	0.17%	50	0.1%	0.13%	0.09%	23
UPC-II	10%	13.54%	4.23%	1177	10%	14.64%	4.58%	1208
	2%	4.63%	1.7%	398	2%	3.15%	1.26%	277
	1%	2.02%	0.62%	166	1%	1.35%	0.48%	115
	0.2%	0.48%	0.27%	47	0.2%	0.29%	0.16%	28
	0.1%	0.29%	0.16%	28	0.1%	0.13%	0.06%	12
UPC-III	10%	13.3%	4.79%	1576	10%	14.12%	5.2%	1683
	2%	4.4%	2.02%	559	2%	3.06%	1.42%	391
	1%	2.08%	0.68%	240	1%	1.4%	0.49%	165
	0.2%	0.42%	0.3%	63	0.2%	0.22%	0.21%	37
	0.1%	0.21%	0.21%	36	0.1%	0.14%	0.13%	23
UPC-IV	10%	14.8%	3.89%	2573	10%	15.92%	4.11%	2757
	2%	5%	1.87%	946	2%	3.72%	1.37%	700
	1%	2.7%	0.84%	487	1%	1.86%	0.7%	353
	0.2%	0.68%	0.37%	145	0.2%	0.41%	0.28%	94
	0.1%	0.36%	0.25%	85	0.1%	0.18%	0.13%	43
UPC-V	10%	14.49%	4.01%	1785	10%	15.71%	4.12%	1914
	2%	4.89%	2.23%	687	2%	3.41%	1.66%	489
	1%	2.34%	0.54%	317	1%	1.61%	0.7%	223
	0.2%	0.58%	0.28%	83	0.2%	0.29%	0.19%	46
	0.1%	0.27%	0.19%	44	0.1%	0.11%	0.1%	21

Table 5.12: TAPS performance on each trace using *Sample and Hold*. TS stands for total scanners detected

Trace	% pkts	SR	FPR	TS	% flows	SR	FPR	TS
UPC-I	10%	10.95%	1.63%	1306	10%	16.85%	1.9%	1947
	2%	3.20%	0.66%	399	2%	3.18%	0.65%	397
	1%	1.89%	0.43%	243	1%	1.91%	0.44%	244
	0.2%	0.59%	0.22%	84	0.14%	%	0.06%	21
	0.1%	0.42%	0.17%	62	0.1%	0.14%	0.06%	21
UPC-II	10%	6.46%	1.51%	501	10%	11%	2.36%	839
	2%	1.81%	0.43%	139	2%	1.77%	0.41%	137
	1%	0.78%	0.23%	65	1%	0.81%	0.25%	67
	0.2%	0.30%	0.13%	27	0.2%	0.02%	0.05%	4
	0.1%	0.19%	0.11%	19	0.1%	0.02%	0.05%	4
UPC-III	10%	5.96%	1.31%	633	10%	10.18%	2.4%	1096
	2%	1.59%	0.45%	180	2%	1.62%	0.51%	185
	1%	0.93%	0.4%	110	1%	0.84%	0.37%	105
	0.2%	0.17%	0.1%	24	0.2%	0.03%	0.02%	5
	0.1%	0.17%	0.1%	17	0.1%	0.03%	0.02%	5
UPC-IV	10%	12.05%	2.38%	1985	10%	18.59%	3.28%	3010
	2%	3.8%	1%	661	2%	3.79%	0.97%	654
	1%	2.31%	0.64%	379	1%	2.1%	0.63%	376
	0.2%	0.63%	0.26%	123	0.2%	0.14%	0.07%	29
	0.1%	0.41%	0.18%	81	0.1%	0.14%	0.07%	29
UPC-V	10%	9.31%	2.32%	1122	10%	15.57%	3.19%	1810
	2%	2.97%	1.09%	377	2%	2.83%	1.04%	373
	1%	1.59%	0.6%	214	1%	1.64%	0.62%	218
	0.2%	0.46%	0.17%	60	0.2%	0.09%	0.05%	14
	0.1%	0.32%	0.13%	44	0.1%	0.09%	0.05%	14

Table 5.13: TAPS performance on each trace using *Smart Sampling*. TS stands for total scanners detected

Trace	% pkts	SR	FPR	TS	% flows	SR	FPR	TS
UPC-I ,	10%	31.14%	2.07%	3449	10%	57.86%	2.96%	6317
	2%	31.12%	2.05%	3445	2%	40.68%	2.55%	4490
	1%	31.11%	2.0%	3444	1%	35.78%	2.3%	3955
	0.2%	31.1%	2.06%	3445	0.2%	31.82%	2.12%	3525
	0.1%	31.1%	2.06%	3445	0.1%	31.5%	2.09%	3489
UPC-II	10%	35.15%	9.05%	2278	10%	49.92%	9.75%	3750
	2%	35.14%	9.05%	2777	2%	40.28%	9.63%	3136
	1%	35.14%	9.05%	2777	1%	37.49%	9.6%	2959
	0.2%	35.4%	9.05%	2777	0.2%	35.69%	9.18%	2820
	0.1%	35.4%	9.05%	2777	0.1%	35.36%	9.07%	2792
UPC-III	10%	29.12%	7.08%	3154	10%	41.92%	7.63%	4317
	2%	29.09%	7.07%	3150	2%	33.31%	7.5%	3555
	1%	29.09%	7.07%	3150	1%	31%	7.36%	3342
	0.2%	29.09%	7.07%	3150	0.2%	29.33%	7.13%	3176
	0.1%	29.09%	7.07%	3150	0.1%	29.2%	7.12%	3164
UPC-IV	10%	31.63%	5.99%	5177	10%	61.71%	6.69%	9413
	2%	31.55%	5.96%	5162	2%	43.26%	6.39%	6834
	1%	31.53%	5.97%	5160	1%	37.28%	6.23%	5989
	0.2%	31.53%	5.97%	5160	0.2%	32.56%	6.07%	5317
	0.1%	31.53%	5.97%	5160	0.1%	32.03%	6.01%	5235
UPC-V	10%	32.38%	8.29%	3924	10%	57.91%	8.78%	6435
	2%	32.36%	8.27%	3920	2%	41.59%	8.59%	4842
	1%	32.36%	8.27%	3919	1%	36.55%	8.45%	4342
	0.2%	32.36%	8.27%	3920	0.2%	32.86%	8.29%	3971
	0.1%	32.36%	8.27%	3920	0.1%	32.52%	8.28%	3937

Table 5.14: TAPS performance on each trace using *Selective Sampling*. TS stands for total scanners detected

Trace	% pkts	SR	FPR	TS	% flows	SR	FPR	TS
UPC-I	10%	99.94%	1.74%	14460	10%	7.07%	0.05%	1013
	2%	23.37%	0.27%	3363	2%	1.61%	0.04%	235
	1%	9.30%	0.10%	1337	1%	0.01%	0%	1
	0.2%	4.82%	0.17%	710	0.2%	0%	0%	0
	0.1%	0%	0%	228	0.1%	0%	0.01%	1
UPC-II	10%	92.41%	3.68%	6038	10%	45.81%	10.65%	3548
	2%	53.41%	11.08%	4052	2%	36.41%	9.26%	2870
	1%	45.23%	10.61%	3509	1%	36.68%	9.31%	2890
	0.2%	37.35%	9.45%	2941	0.2%	35.39%	9.02%	2791
	0.1%	36.17%	9.31%	2858	0.1%	35.2%	9.07%	2782
UPC-III	10%	92.37%	9.62%	8885	10%	37.82%	8.64%	4048
	2%	44.33%	9.48%	4688	2%	30.07%	7.3%	3256
	1%	37.14%	8.48%	3975	1%	30.29%	7.23%	3269
	0.2%	30.79%	7.35%	3322	0.2%	29.22%	7.12%	3166
	0.1%	29.97%	7.27%	3244	0.1%	29.13%	7.09%	3156
UPC-IV	10%	85.66%	2.98%	12198	10%	42.93%	7.14%	6890
	2%	50.18%	7.35%	7918	2%	32.6%	6.19%	5338
	1%	42.31%	6.96%	6781	1%	33.36%	6.22%	5447
	0.2%	33.22%	6.22%	5428	0.2%	31.85%	5.98%	5206
	0.1%	32.39%	6.18%	5308	0.1%	31.65%	5.97%	5178
UPC-V	10%	87.1%	3.87%	8777	10%	43.02%	9.34%	5052
	2%	50.16%	9.7%	5776	2%	33.24%	8.31%	4009
	1%	42.51%	9.29%	4998	1%	33.99%	8.52%	4102
	0.2%	33.98%	8.28%	4078	0.2%	32.58%	8.32%	3947
	0.1%	33.07%	8.25%	3987	0.1%	32.41%	8.28%	3926

Chapter 6

Online Selective Sampling

6.1 Motivations

There are two main motivations to build this new sampling technique. In the first place, many anomalies (portscans, DDoS, worms) use small flows and therefore we are interested in keeping them rather than the large ones in order to easily detect the attacks. There is already a sampling mechanism which goal is to take just small flows (explained in Chapter 3). The problem of *Selective Sampling* [27] is that it first needs to capture all the packets. After that, it proceeds by sampling entire flows. The main issue of working that way is that all the flows must be saved into memory, thus being contradictory with the main goal of sampling, which is to save resources by just keeping a portion of all the traffic. The goal of *SES* is to reduce the used bandwidth between the collector and the recollector. Hence, our proposal has the same goal (sample small flows) but without capturing all the traffic, thus requiring significantly less resources.

6.2 Ideas and background

Before starting the explanation we will review first how the other technique that has the same goal works. Androulidakis et al. used the following metrics:

- z: maximum size for a flow to be considered small ($z \geq 1$).
- c: probability of sampling a small flow. ($0 < c \leq 1$).
- n: parameter used to reduce the amount of non-small flows sampled ($n \geq 1$). Large flows sampling probability is directly related to their size (the more packets the lower is the probability). n allows to further regulate that sampled flows percentage.

Selective Sampling [27] samples a flow of x packets according to the following expression:

$$p(x) = \begin{cases} c & x \leq z \\ \frac{z}{n \cdot x} & \text{otherwise} \end{cases}$$

In order to proceed with an accurate comparison among both sampling methods, the idea is to build our sampling technique as equivalent to *SES* as possible but without capturing all the traffic. To achieve that goal we need a mechanism that allows us to decide either to keep or discard an incoming packet in a rapid way. The idea is to sample packets from a flow while this flow continues being small. For instance, if we define a small flow as a n-packet flow ($n \geq 1$), our sampling method is going to sample all flows having at most n packets with a certain probability and discard all the flows having $n + 1$ packets or more with a probability directly related to their size (the more packets, the higher should be probability of not taking that flow).

The simplest solution used to track flows is the hash table approach. However, nowadays it is becoming impracticable due to high-speed links. Each flow identifier must be keeped and many memory accesses are necessary for each incoming packet. In addition to that, collisions and new flow entries must be taken into account. The inter arrival times in links of several Gb/s are of the order of nanoseconds, thus requiring the process time per packet to be really short. We finally decided to use bloom filters because the time of looking for an element is very fast.

6.2.1 Bloom filters

The idea of a bloom filter [35] is to provide a fast way to decide if an element belongs to a given set. In our case, it is used to control whether a certain flow had been discarded until now. If it it has been dropped before, the packet will be directly discarded. Otherwise, the packet belonging to that flow will be processed normally. If taking the new packet into account makes that its flow is unsampled, that flow is immediately set into the bloom filter.

A bloom filter uses a bitmap, which is an array of bits of size m. Initially, all the positions of that bitmap are set to 0. When an element arrives (a new packet in our case) a certain number of hash functions (k) are applied to it. Since we are tracking flows we apply those functions to the corresponding 5-tuple of the packet. Every position referenced by the hash functions is set to 1. According to the expected different elements (flows) to count (n) and the desired false positive probability (p), k and m are set. In order to know if a given element has been already seen, all the positions referenced by its hash functions must have value 1. Otherwise, it means that the element is new. It may be possible (depending on

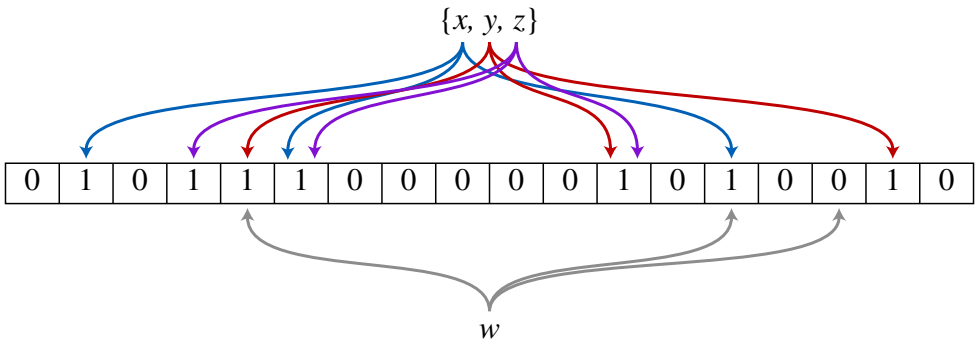


Fig. 6.1: An example of a Bloom filter, representing the set x, y, z. The colored arrows show the positions in the bit array that each set element is mapped to. The element w is not in the set, because it hashes to a position of the bitmap which has value 0

p) to obtain an incorrect answer after looking for an element in the bitmap (false positive). There are no false negatives. We can observe an example in Figure 6.1 in which $m = 18$, $k = 3$ and $n = 4$.

[35] explains exactly how to dimension the bloom filter according to the desired false positives and the number of different elements to count. A lot of tools like the one we chose [36] use that formulas to do these computations and make the bloom filter configuration easier.

6.3 Our sampling method

In this section, we present the working scheme of our new sampling technique called *Online Selective Sampling (OSES)*.

6.3.1 How it works

OSES working scheme can bee seen on Algorithm 1. For every incoming packet we first check if its corresponding flow has been previously discarded by looking at the bloom filter (line 2). If there is a match, the packet is discarded (line 3), because it means that the corresponding flow has been already unsampled before.

If the element is not in the bloom filter (line 4), it means that, until now, the flow which that packet belongs to has not been dropped yet. Now we must check if this condition still holds taking the new packet into account:

- If it does not (line 8), we take the packet with a certain probability q (line 10). This probability is deeply explained in Section 6.3.2 and Algorithm 2. If the

packet turns out to be discarded (line 14), its flow is deleted from the hash table (line 12) and its 5-tuple is set in the bloom filter (line 13).

- Otherwise (line 17), the packet is sampled with probability c.

6.3.2 Correcting the probability

Since we do not capture the entire flow, we do not know its final size. We only know its size until the current packet. Since the goal is to sample the same flows that *Selective Sampling* but without capturing all the traffic, we must find a way to make that the probability of taking a flow packet by packet would be exactly the same that in the case of sampling the flow directly when all its packets have been captured. If that occurs, both sampling methods would be equivalent. In order to make this happen, the probability for *OSES* must be corrected at each step in such a way that the addition of probabilities until the last packet results in exactly the same probability as directly sampling that flow with *SES*. Our method should do what *SES* does but step per step (packet per packet).

For instance, suppose that we have a flow of size $x = 5$ packets (previous full traffic capture needed). We configure *SES* with $c = 0.9$, $n = 1$ and $z = 2$. According to *SES* formula and these parameters, since $5 > 2$ the probability of taking that flow is the following:

$$\frac{2}{1 \times 5} = 0.40$$

Going back to *Online Selective Sampling* and considering the same scenario, we will see how our method should proceed. In order to reach the 5th packet, the previous packets must be sampled too, because otherwise, the packets of that flow would be directly discarded (because the flow would have been set in the bloom filter). We have to take into account that the events are not independent. For instance, if the 3rd packet is discarded, both the 4th and the 5th packets will be directly dropped as well. Thus, in order to reach the last packet of a flow (the 5th in this case), all the previous packets must be sampled too. The probability computed for each packet must be adjusted according to the previous packets that have been taken to make it possible to reach the present packet. Therefore, the final sequence from the first to the last packet of the flow should be the following. In each step we can see how probability is corrected by previous packets:

1. Corrected probability for packet 1:
   - $cp(1) = 1 - c = 0.1$

2. Corrected probability for packet 2:
    - $cp(2) = 1 - c - cp(1) = 0.1 - 0.1 = 0$

3. Corrected probability for packet 3:
    - $cp(3) = (1 - 0.66) - cp(2) - cp(1) = 0.23$

4. Corrected probability for packet 4:
    - $cp(4) = (1 - 0.5) - cp(3) - cp(2) - cp(1) = 0.17$

5. Corrected probability for packet 5:
    - $cp(5) = (1 - 0.4) - cp(4) - cp(3) - cp(2) - cp(1) = 0.10$

These calculations are possible because the probability of dropping a flow is always bigger for the actual packet than for the former. This occurs because the algorithm gives priority to small flows: the higher is the flow size, the lower is the probability of sampling it (the discarding probability always grows). According to the corrected probability for each packet (above) we can see below what is the real probability of not taking each packet at every step:

1. Probability of not taking it:
    - $p(1) = cp(1) = 0.1$

2. Probability of not taking it:
    - $p(2) = cp(2) + cp(1) = 0.1$

3. Probability of not taking it:
    - $p(3) = cp(3) + cp(2) + cp(1) = 0.33$

4. Probability of not taking it:
    - $p(4) = cp(4) + cp(3) + cp(2) + cp(1) = 0.5$

5. Probability of not taking it:

   - $p(5) = cp(5) + cp(4) + cp(3) + cp(2) + cp(1) = 0.60$

$p(5) = 0.6$, which matches exactly with the probability of discarding an entire flow of 5 packets directly with *Selective Sampling* calculated before: $1 - 0.40 = 0.6$. Generalising the 5-packet flow to an y-packet flow, the probability of discarding it is the following:

$$p(y) = \sum_{i=1}^{y} cp(i)$$

$$cp(y) = \begin{cases} 0 & y = 0 \\ (1-c) - \sum_{i=1}^{y-1} cp(i) & y \leq z \text{ and } y > 0 \\ (1 - \frac{z}{n \times y}) - \sum_{i=1}^{y-1} cp(i) & \text{otherwise} \end{cases}$$

As we can see it is a recursive definition because the probability at each step must be corrected by the previous ones. Algorithm 2 shows the implementation.

6.3.3 Constraints

As already mentioned before, this step by step computations work because the discard probability increases with the flow size (except while the flow has up to z packets, in which case is always $1 - c$). If that condition did not decrease, *OSES* would return negative probabilities. That happens if the following expression is not accomplished:

$$c \geq \frac{z}{n \times (z+1)}$$

Why this condition must hold?

The probability of discarding a flow of $(z+1)$-packets is the following:

$$p(z+1) = \sum_{i=1}^{z+1} cp(i)$$
$$cp(z+1) = \left(1 - \frac{z}{n \times (z+1)}\right) - \sum_{i=1}^{z} cp(i)$$

In order to have that $cp(z+1) \geq 0$, the below expression must hold:

$$1 - \frac{z}{n \times (z+1)} \geq \sum_{i=1}^{y-1} p(i)$$

If z is odd $\sum_{i=1}^{z} p(i) = 1 - c$. Otherwise, $\sum_{i=1}^{z} p(i) = 0$. Therefore:

$$\left(1 - \frac{z}{n \times (z+1)}\right) \geq (1 - c) \text{ or } \left(1 - \frac{z}{n \times (z+1)}\right) \geq 0$$

In the first formula the following expression is equivalent:

$$\left(1 - \frac{z}{n \times (z+1)}\right) \geq (1 - c) \Rightarrow c \geq \frac{z}{n \times (z+1)}$$

In the second case it is obvious that the condition holds, because $z < z + 1$ and $n \geq 1$, thus making the fraction being a number minor than 1. Therefore, the following substract will be positive for sure.

Although this extra constraint, we consider that since *Selective Sampling* is basically used to sample small flows, it does not make to much sense to sample small flows with higher probability than the non-small. It is a little bit contradictory to use either *OSES* or *SES* violating that condition.

6.3.4 Preliminary results and remaining issues

As we expected, our first main goal seems to be accomplished. Preliminary results showed that both methods (configured with the same parameters) sampled approximately an equal fraction of flows and packets. The other principal objective that we had (save resources in comparison to the full capture of *SES*) looked like being achieved as well: *Online Selective Sampling* used significantly less memory while sampling approximately the same portion of the traffic.

Since it is equivalent to *Selective Sampling* and given that during our study, the former sampling method exhibited a remarkable performance for both TRW and TAPS (see Chapter 5), it would mean that we would be able to achieve acceptable behaviour for both portscan detection mechanisms under sampling on a per-packet sampling basis, thus being implementable in Sampled NetFlow. However, further evaluation and a complete analysis are still needed.

So far, we have concluded that *Online Selective Sampling* and *Selective Sampling* are equivalent with the only difference that our method will sample traffic in real-time, without having to capture it all before. But there is a remaining problem that needs to be further studied in order to determine its impact on *Online Selective Sampling* accuracy: the false positives of the bloom filter. It is possible that we directly drop a packet from an unseen flow until now. It could happen if when we look for it in the bloom filter, the bloom filter returns a match when the truth is that it is a false positive.

Algorithm 1: Online Selective Sampling Algorithm

Input: BF: bloom filter;
HT: Hash table of packets;
z: it defines a small flow (in packets);
c: it defines the probability of sampling a small flow;

```
1  for every incoming packet pkt do
2      if get(BF, pkt) then
3          discard pkt;
4      else
5          flow = lookup(HT, get_tuple(pkt));
6          r = random();
7          x = size_packets(flow);
8          if x > z then
9              /* probability of not taking the packet (deeply explained in next
                   section and Algorithm 2) */
10             q = 1 − p(x);
11             if q > r then
12                 delete(HT, flow);
13                 set(BF, pkt);
14                 discard pkt;
15             else
16                 sample pkt;
17         else
18             if c < r then
19                 delete(HT, flow);
20                 set(BF, pkt);
21                 discard pkt;
22             else
23                 sample pkt;
```

Algorithm 2: probability_BF

Input: $pkts\_flow$: packets of the flow until now;
n: Parameter that further reduce the fraction of non-small flows sampled;
c: Parameter that indicates the probability of sampling small flows
z: it defines a small flow (in packets);

```
1  if pkts_flow == 0 then
2      return 0;
3  else
4      if pkts_flow ≤ threshold then
5          q = 1 − c;
6      else
7          q = 1 − (z/(n × pkts_flow));
8      i = pkts_flow − 1;
9      while i ≥ 1 do
10         /* recursive call */
11         q = q − probability_BF(i);
12         i = i − 1;
13     return q;
```

Chapter 7

Conclusions and Future Work

In this technical report, we have analyzed the impact of sampling on two portscan detection algorithms to test whether they are robust enough to continue finding portscans reliably. In contrast to some previous studies, we have been able to see that that, sometimes, *Packet Sampling* exhibited better than *Flow Sampling* for portscan detection. Furthermore, given that our study indicated the impact of sampling on both portscan detection methods performance was huge, we proposed a modification of an already existing sampling technique that showed the best results but that was not NetFlow compatible. Preliminary results, indicate that our sampling technique samples approximately the same traffic that the already existing mechanism while significantly reducing memory usage. Moreover, it is implementable in NetFlow.

Regarding the analyzed portscan detection algorithms, we observed that both TRW and TAPS were dramatically affected by sampling. However, even that TRW turned out to be almost useless under sampling, TAPS exhibited a more acceptable performance, thus being the best choice for portscan detection under sampling.

Concerning to the sampling techniques, while *Flow Sampling* and *Packet Sampling* exhibited similar performance using TRW, with TAPS we observed that *Packet Sampling* clearly outperformed it when using the same fraction of packets. *Smart Sampling* showed to be useless under TRW but, it exhibited a remarkable performance under TAPS. *Sample and Hold* did not perform well for any sampling technique and portscan detection method. Finally, *Selective Sampling*, presented the best behaviour among all the sampling techniques achieving acceptable performance for both TRW and TAPS.

The results presented in this work are not entirely aligned with those obtained in former studies. In particular, it has been previously concluded that *Flow Sampling* was always the most promising sampling method to detect portscans, but according to our results, we have not observed this superiority in all the experi-

ments, thus confirming that this parallel study reveals new interesting information.

Taking into account that *Selective Sampling* showed to be the best sampling mechanism but it do not sample on a per-packet basis and we are interested in NetFlow, we propose a NetFlow compatible equivalent sampling technique. It is called *Online Selective Sampling* and it does not need to capture all the traffic before sampling because it works taking decisions packet by packet. Preliminary results showed that our sampling technique samples the same traffic that *Selective Sampling* while saving a lot of memory. Nonetheless, we are still working on it and further testing of the method remain pending.

We are currently analyzing TAPS performance degradation in more detail as well as further testing our sampling technique proposal. Moreover, we are also working on an entropy-based portscan detection technique. We plan to extend this study to other sampling techniques and anomalies.

Part of the results and conclusions of this work resulted in an article published in the proceedings of the *First International Workshop on Traffic Monitoring and Analysis* placed in Aachen, Germany on 11th May 2009 [37].

Bibliography

[1] Cisco Systems, "NetFlow services and applications," White Paper, 2000.

[2] ——, "Sampled NetFlow," http://www.cisco.com/en/US/docs/ios/12_0s/feature/guide/12s_sanf.html.

[3] M. Roesch, "Snort–lightweight intrusion detection for networks," in *Proc. of USENIX Systems Administration Conference*, 1999.

[4] J. Jung, V. Paxson, A. Berger, and H. Balakrishnan, "Fast portscan detection using sequential hypothesis testing," in *Proc. of IEEE Symposium on Security and Privacy*, 2004.

[5] S. Avinash, T. Ye, and B. Supratik, "Connectionless portscan detection on the backbone," in *Proc. of IEEE International Performance Computing and Communications Conference*, 2006.

[6] V. Paxson, "Bro: a system for detecting network intruders in real-time," *Computer Networks*, vol. 31, no. 23-24, 1999.

[7] J. Mai, A. Sridharan, C. Chuah, H. Zang, and T. Ye, "Impact of packet sampling on portscan detection," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 12, 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 conference on Internet measurement*, 2006.

[9] D. Brauckhoff, B. Tellenbach, A. Wagner, M. May, and A. Lakhina, "Impact of packet sampling on anomaly detection metrics," in *Proc. of ACM SIGCOMM conference on Internet measurement*, 2006.

[10] P. Barford and D. Plonka, "Characteristics of network traffic flow anomalies," in *Proc. of ACM SIGCOMM Workshop on Internet Measurement*. ACM New York, NY, USA, 2001, pp. 69–73.

[11] J. Brutag, "Aberrant behavior detection and control in time series for network monitoring," in *Proc. of the 14th Systems Administration Conference (LISA)*, 2000.

[12] P. Barlet-Ros, J. Solé-Pareta, J. Barrantes, E. Codina, and J. Domingo-Pascual, "SMARTxAC: a passive monitoring and analysis system for high-speed networks," *Campus-Wide Information Systems*, vol. 23, no. 4, 2006.

[13] P. Barlet-Ros, , H. Pujol, J. Barrantes, J. Solé-Pareta, and J. Domingo-Pascual, "A system for detecting network anomalies based on traffic monitoring and prediction," *Boletín de RedIRIS*, vol. 74-75, no. 1, pp. 23–27, December 2005, (in Spanish).

[14] H. Wang, D. Zhang, and K. Shin, "Change-point Monitoring for the detection of DoS attacks," *IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING*, pp. 193–208, 2004.

[15] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," in *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2004, pp. 219–230.

[16] A. Lakhina, K. Papagiannaki, M. Crovella, C. Diot, E. Kolaczyk, and N. Taft, "Structural analysis of network traffic flows," in *Proceedings of the joint international conference on Measurement and modeling of computer systems*. ACM New York, NY, USA, 2004, pp. 61–72.

[17] A. Lakhina, M. Crovella, and C. Diot, "Characterization of network-wide anomalies in traffic flows," in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*. ACM New York, NY, USA, 2004, pp. 201–206.

[18] ——, "Mining anomalies using traffic feature distributions," in *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM New York, NY, USA, 2005, pp. 217–228.

[19] P. Barford, J. Kline, D. Plonka, and A. Ron, "A signal analysis of network traffic anomalies," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*. ACM New York, NY, USA, 2002, pp. 71–82.

[20] K. Xu, Z. Zhang, and S. Bhattacharyya, "Profiling internet backbone traffic: behavior models and applications," in *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM New York, NY, USA, 2005, pp. 169–180.

[21] P. Brockwell, R. Davis, and I. NetLibrary, "Introduction to time series and forecasting," 2002.

[22] C. Lee, C. Roedel, and E. Silenok, "Detection and characterization of port scan attacks," *Univeristy of California, Department of Computer Science and Engineering*, 2003.

[23] J. Mirkovic, J. Martin, and P. Reiher, "A Taxonomy of DDoS Attacks and DDoS Defense Mechanisms," 2001.

[24] M. Walfish, M. Vutukuru, H. Balakrishnan, D. Karger, and S. Shenker, "DDoS defense by offense," in *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM New York, NY, USA, 2006, pp. 303–314.

[25] G. Androulidakis, V. Chatzigiannakis, S. Papavassiliou, M. Grammatikou, and V. Maglaris, "Understanding and evaluating the impact of sampling on anomaly detection techniques," in *Military Communications Conference, 2006. MILCOM 2006*, 2006, pp. 1–7.

[26] G. Androulidakis, V. Chatzigiannakis, and S. Papavassiliou, "Using selective sampling for the support of scalable and efficient network anomaly detection," *Globecom Workshops, 2007 IEEE*, pp. 1–5, 2007.

[27] G. Androulidakis and S. Papavassiliou, "Intelligent flow-based sampling for effective network anomaly detection," in *Global Telecommunications Conference, 2007. GLOBECOM'07. IEEE*, 2007, pp. 1948–1953.

[28] V. Chatzigiannakis, S. Papavassiliou, G. Androulidakis, and B. Maglaris, "On the realization of a generalized data fusion and network anomaly detection framework," in *Fifth International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP 06)*. Patra, Greece, July 2006.

[29] IETF Packet Sampling (PSAMP) Working Group, http://www.ietf.org/html.charters/psamp-charter.html.

[30] Sampling and Filtering Techniques for IP Packet Selection (RFC 5475), http://www.ietf.org/rfc/rfc5475.txt.

[31] N. Duffield, "Sampling for passive internet measurement: A review," *Statistical Science*, vol. 19, no. 3, 2004.

[32] C. Estan and G. Varghese, "New directions in traffic measurement and accounting: focusing on the elephants, ignoring the mice," *ACM Transactions on Computer Systems*, vol. 21, no. 3, 2003.

[33] N. Duffield, C. Lund, and M. Thorup, "Properties and prediction of flow statistics from sampled packet streams," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*. ACM New York, NY, USA, 2002, pp. 159–171.

[34] IST-Lobster sensor at UPC, http://loadshedding.ccaba.upc.edu/appmon.

[35] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, 1970.

[36] Bloom Filter Calculator, http://hur.st/bloomfilter.

[37] I. Paredes-Oliva, P. Barlet-Ros, and J. Solé-Pareta, "Portscan detection with sampled netflow," in *Proceedings of International Workshop on Traffic Monitoring and Analysis*, ser. Lecture Notes in Computer Science, vol. 5537. Springer, May 2009, pp. 26–33. [Online]. Available: http://www.springerlink.com/content/k57772t5641w5023/