RelSamp: Preserving Application Structure in Sampled Flow Measurements

Myungjin Lee, Mohammad Hajjat, Ramana Rao Kompella, and Sanjay Rao
Purdue University

***Abstract*—The Internet has significantly evolved in the number and variety of applications. Network operators need mechanisms to constantly monitor and study these applications. Given modern applications routinely consist of several flows, potentially to many different destinations, existing measurement approaches such as Sampled NetFlow sample only a few flows per application session. To address this issue, in this paper, we introduce RelSamp architecture that implements the notion of *related sampling* where flows that are part of the same application session are given higher probability. In our evaluation using real traces, we show that RelSamp achieves 5-10x more flows per application session compared to Sampled NetFlow for the same effective number of sampled packets. We also show that behavioral and statistical classification approaches such as BLINC, SVM and C4.5 achieve up to 50% better classification accuracy compared to Sampled NetFlow, while not breaking existing management tasks such as volume estimation.**

I. Introduction

The tremendous success of the Internet, and the fertile ground for innovation that it provides has spawned a diverse range of applications, with new applications continually and rapidly emerging and gaining in prominence. The last decade alone has seen rapid growth in popularity of peer-to-peer (p2p) systems (*e.g.*, BitTorrent), online social networks (*e.g.*, Facebook), online gaming applications (*e.g.*, World of Warcraft), and so on. While Internet traffic was predominantly dominated by the Web in the 1990's, p2p traffic accounted for over 60% of traffic around 2005, and more recently, video-based applications such as YouTube are gaining in popularity. Concurrent with the growth of new applications and changes in popularity across applications, we are continually seeing shifts in characteristics and communication patterns of existing applications. The emergence of new applications, and their rapidly changing characteristics require network operators to continuously measure and monitor traffic characteristics in their networks. These measurements allow operators to, for instance, potentially re-provision their networks, detect any new types of undesirable behavior within applications, and in general, prepare their networks better against any major application trends.

Now, consider a router at the edge of a large-scale campus network or an enterprise network—typically referred to as a *gateway*—where a network operator is interested in collecting measurements. Typical enterprise gateway routers today operate at 1-10 Gbps capacity and these line rates are poised to increase further as technology scales. To monitor traffic at these high rates, one could potentially use a high-speed packet capture card (*e.g.*, DAG cards made by Endace [1]). There are two problems however: First, these capture cards tend to be quite costly—a 10Gbps capture card alone costs around $20-30K. Second, the volume of data one needs to collect is huge (a 1 TeraByte disk will be filled up in approximately 2 minutes). Thus, most network operators prefer to collect *flow records* provided by NetFlow supported by many modern routers. Because collecting *all* flow records is also quite difficult, operators use Sampled NetFlow which essentially computes flow records on a sampled stream of packets. (At 10 Gbps, a 1-in-1024 sampling rate is typically used.)

Unfortunately, Sampled NetFlow fails to preserve application structure that makes it ill-suited to monitor the new range of applications evolving in the Internet today. In particular, many emerging Internet applications (*e.g.*, p2p or cloud-based services) are routinely composed of many different flows to potentially different servers/hosts that are often geographically distributed. With random packet sampling, only a small subset of flows, if any, are sampled for an application session. The impact of this effect is two fold: First, application structure becomes distorted in the sampled data. For example, assume we are interested in the size of p2p applications in terms of their network footprint (number of servers contacted, number of flows, etc.). Reconstructing the application characteristics from Sampled NetFlow records is clearly hard. Second, application classification from sampled data becomes harder. Several researchers have already pointed out the inadequacies of simple port-based classification for emerging applications such as p2p [2], [3], [4]. Several alternate approaches based on statistical techniques, or host behavioral patterns have emerged [3], [5], [6], [4], [7], but most of this work has dealt with flow records computed from unsampled data. Since these techniques may rely on traffic features across flows (*e.g.*, BLINC [4] matches the communication behavior in flow records with specific graphlets), and sampling distorts flow correlations, application classification becomes harder with sampled flow records.

Motivated by these limitations of random packet sampling, in this paper, we propose the notion of *related sampling* based on the following key idea: *Once a flow is sampled, all flows that are part of the same application session, are sampled with high probability.* If we set this higher sampling probability to 1, then *all* flows part of the application session will be sampled. The downside is that all the available processing and memory budget will be devoted to a smaller number of application sessions. One could therefore consider choosing to sample these related flows with a more modest 10% or 20%

probability. This will have the effect of sampling more number of application sessions with slightly less fidelity (compared to the 100% case) representing a trade-off between number of application sessions and measurement fidelity.

In this paper, we propose an architecture called RelSamp to explore the potential of related sampling. Ideally, flows corresponding to the same application session must be identified as related. However, since determining this is hard, RelSamp considers all flows that contain the same source IP address created within a given amount of time from each other as related. This heuristic is motivated from a measurement study on a 13-hour campus trace. The RelSamp architecture incorporates related sampling with the help of three stages of sampling. First, we use a host selection probability that controls which host gets selected for subsequent packet selection. Once a host gets selected, packets are subject to a flow-selection probability that governs the probability with which a flow that contains the host as the source IP address is created. Finally, the last stage of packet sampling dictates the rate at which flow records are updated. Thus, RelSamp biases packet and flow selection in favor of hosts that are already admitted.

Thus, the paper has the following main contributions:

First, we introduce *related sampling* that allows flows that are part of the same application session to be sampled with higher probability. Our architecture allows selecting a large majority of flows from a given application session thus allowing scalable monitoring and characterization of new and emerging Internet applications. Second, using real traces, we extensively evaluate the efficacy of RelSamp. In our results, we observe that RelSamp is capable of obtaining **5-10x** more flows per application session compared to Sampled NetFlow and flow sampling. We also show that the classification accuracy of BLINC, SVM and C4.5 increases by **up to 50%** when RelSamp is used instead of Sampled NetFlow.

II. Understanding Relatedness of Flows

In this section, we first use a measurement study based on real packet traces to understand how to capture the notion of relatedness of flows. We then use this understanding to propose our RelSamp architecture in the next section.

A. Capturing relatedness of flows

Typical measurement solutions operate at the granularity of a flow consisting of the 5-tuple (source and destination IP addresses, source and destination ports, and the protocol field). However, application sessions typically involve many flows, potentially to many different destinations. A Web application session, for instance, consists of the set of flows a host originates in order to download Web objects from different Web servers. Thus, the fundamental unit of measurement in our framework is an *application session*, defined as the set of flows that correspond to *a given application* that originate at a given host to one or many other hosts from a certain time (t) with the maximum packet inter-arrival time (τ).

Figure 1 pictorially represents application sessions each of which is depicted as an oval. For example, App1session1 and App1session2 represent two sessions of the same application. The $x-axis$ shows the time and each arrow represents a flow of a certain application type, originating from the same host.

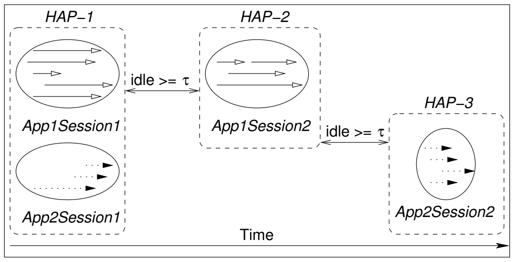


Fig. 1. HAPs vs application sessions. The ovals represent flows within a given application session. The rectangles represent flows within a HAP.

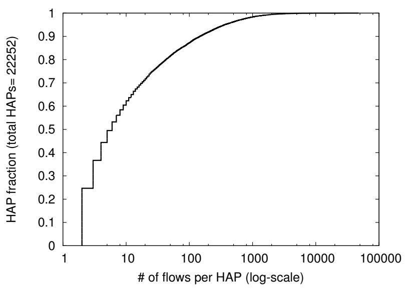
While application sessions provide a useful conceptual framework, it is difficult for routers in the middle of a network to identify flows belonging to the same application session. In recent years, it is no longer feasible to use port numbers to represent applications directly, as many applications routinely use non-standard ports [2], [3], [4]. Deep packet inspection could be used to identify and detect applications, however such techniques are too computationally expensive to be performed in an *online* fashion at routers on high-speed links. While researchers are beginning to look at online classification by, for example, observing a few packets of a TCP session [6], this research is still in its early stages.

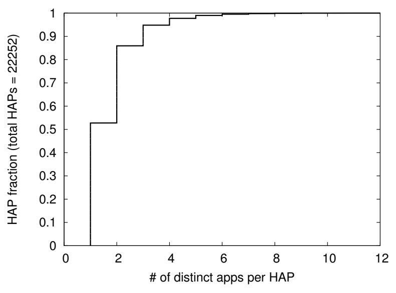
In order to deal with the scalability issues posed by online application classification in routers, we propose the notion of a *host activity period* (HAP), defined as the set of flows that originate at a host to one or more destinations from a certain time (t) with the maximum packet inter-arrival time (τ) regardless of their application class.

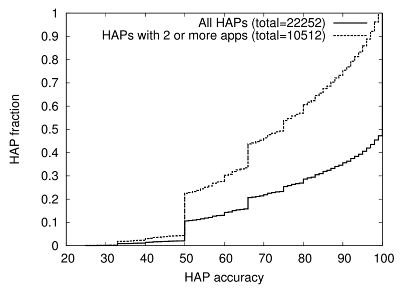
We observe that if a host runs exactly one application, then the corresponding application session and HAP are identical (assuming both definitions use the same threshold value τ). All bursts of flows temporally close to each other originating from the same host are put together in the same HAP. Thus, from Figure 1, HAP-1 (a HAP is represented by a dashed rectangle) encompasses two application sessions, App1session1 and App2session1, while HAP-2 and HAP-3 share a one-to-one correspondence with the actual application sessions.

B. HAPs vs application sessions

We now explore the relationship between HAPs and application sessions using an empirical measurement study. Our study is conducted using a 13-hour packet trace collected within a campus network. The trace contains full payload data, which enables us to identify the applications correctly using deep-packet inspection techniques. We use a τ value of 30 seconds for both the application session as well as HAP definitions for this study. Small values of τ could result in splitting application sessions (or HAPs) in too fine-grained fashion, while large values could cause flows from many applications to be present in the same HAP. We have varied τ between 3 and 600 seconds, and found 30 to be a reasonable choice. We summarize our key findings (the graphs are shown in Figure 2):


(a) Number of flows per HAP


(b) Number of applications per HAP



(c) Accuracy

Fig. 2. Number of flows and applications per HAP.

HAPs can have a large number of flows: We analyze the distribution of number of flows that constitute a HAP in Figure 2(a). We focus on only the set of HAPs that have at least two flows. Out of these, 22% of HAPs consist of exactly two flows. Out of the remaining ones, more than 50% of them have at least 10 flows. About 15% of HAPs consisted of greater than 100 flows indicating that host activity can be quite intensive.

Most HAPs have a small number of applications: We observe in Figure 2(b) that a majority of HAPs consist of a very small number of applications, with almost 50% and 85% of HAPs consisting of only one and two applications, respectively.

For HAPs with more than one application, there typically exists one dominant application: We define the accuracy of a HAP as the percentage of flows that belong to the application that has the most flows in the HAP. Among those HAPs which have more than one application, we observed that more than 80% of the HAPs have an accuracy of greater than 70%.

Implications for RelSamp design: Our results based on this measurement study indicate that the notion of HAPs approximates the application sessions. Further, given that HAPs (application sessions) can have a large number of flows, the results indicate the importance of capturing related flows to get representative characterizations of applications. Note that the existence of a few instances when a HAP may contain multiple application sessions, does not pose explicit problems in our measurement framework. This is since, we can post-process the collected flow records within a HAP by applying classification approaches such as BLINC [4] to split a HAP into multiple application sessions. The efficacy of BLINC, however, depends on how well we capture (most of) the application sessions, which we address with RelSamp next.

III. ARCHITECTURE OF RELSAMP

A. Design

The *key idea* of RelSamp is to create flow records by sampling flows that are related to already sampled flows with higher probability so that more flows within a HAP are collected. This small bias, as we shall show later in our evaluation, is remarkably effective at collecting many more number of flows per application than with Sampled NetFlow, thus facilitating better application classification as well as continuous and ubiquitous characterization of application traffic.

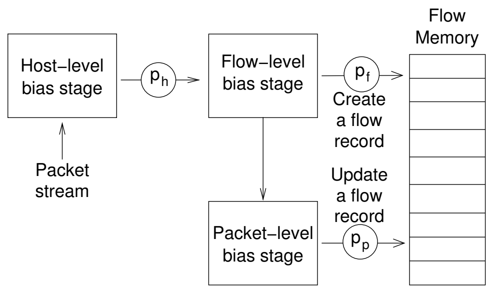


Fig. 3. Architecture of RelSamp.

The basic design of RelSamp is shown in Figure 3, and is very similar to NetFlow in terms of the flow memory and the flow record structure. It mainly differs from Sampled NetFlow in its sampling mechanism. Conceptually, RelSamp consists of three stages of sampling—host selection, flow creation and flow update. The purpose of the host selection stage is to select a HAP for measurement. Once a HAP is probabilistically selected, the flow selection stage determines what percentage of the flows within the HAP to observe. Flows for which records exist are then updated periodically with a flow update probability. The accuracy of flow statistics for each created record is governed by the flow update probability. These three stages together control the total number of sampled packets affecting the CPU utilization, and the number of records created affecting the memory utilization. We now describe the individual stages in more detail.

Host sampling. There are many different architectural options for sampling HAPs. One option is to maintain some amount of state for active HAPs with a hash-table (call it HAP table). If the source IP address of a packet is not already contained in the HAP table, then the host could be sampled with some probability, p_h, and an entry could be created in the HAP table. The advantage is that only HAPs with sufficient volume will be sampled. Unfortunately, the above option has two problems. First, it has to maintain extra state to create and expire HAP table entries, that may be complicated. Second, some number of packets would be missed before flow records are created, that need to be accounted for. These packets may belong to several flows, and it is not clear how to create unbiased estimators for flow records.

Instead, in RelSamp, we sample HAPs using a *hash-based selection* on IP addresses similar to flow-sampling in [8]. In other words, we focus only on a subset of source IP addresses that are selected by hashing these source IP addresses and checking whether the hash lies within a pre-configured range. By controlling the hash range to the total hash space, we can control the host sampling rate. Because hosts are either selected on the first packet or not at all, no packets are missed before a host is selected at the flow-level. Thus, unbiased estimators are easy to create in this framework, as we shall describe in Section III-B. Further, it does not require any additional HAP table state for maintaining HAP entries. HAPs could be easily constructed by post-processing the sampled flow records that contain the start and end timestamps anyway (by checking whether the flows are separated by more than τ). A potential concern with hash-based selection is that, only certain hosts will be selected, while others may not be sampled at all. We can easily solve this by changing the hash function periodically. Alternately, we can choose higher sampling probabilities for p_h to minimize its impact.

Flow creation. Once a particular host is selected based on the hash-based host selection, the packet is handed to the next stage where a flow is created for the packet, if it does not exist already, with a probability p_f. This stage presents network operators with the flexibility to choose what percentage of flows for a given host are selected. At this stage, again one can choose either hash-based or packet-based selection. Packet based selection creates flow records for heavy-hitter flows, while hash-based selection will create flow records for all types of flows. We pick packet-based selection since volume estimates are more accurate.

Flow update. The final stage in our RelSamp architecture is the flow update stage. Packets for flows that are already existing in the flow cache are updated with probability p_p. This gives an operator additional flexibility to specify the accuracy with which individual flow records are updated. In Sampled NetFlow, the flow creation and the flow update probabilities, p_f and p_p are equal to the configured sampling probability. The reason we split this base probability into two parts is to provide network operators the ability to trade-off accuracy of each flow with more number of flows per host. Thus, for a given *effective sampling rate* p_e (that dictates how many effective packets are sampled), we can choose to provide a higher p_f that allows creating more number of flows, each of which is updated with a lower sampling probability p_p.

For the purposes of increasing related application flows, which is the main goal of our architecture, we need to increase the number of flows that share common source IP address (by setting p_f to be high). Of course, we still need to ensure that the total number of packets lies within a given packet sampling budget, p_e, which naturally requires configuring the value of either p_p or p_h to be small. The effect of reducing p_p is that individual flow statistics may suffer from higher errors. The effect of reducing p_h is that aggregate volume estimates become more inaccurate, as scaling the volumes from the observed hosts to the actual hosts becomes skewed as number of hosts decreases. Thus, there is a natural trade-off among these three variables, that need to be configured depending on the objectives and the particular location at which sampling is being performed. We evaluate this trade-off empirically in Section IV using a real 10 Gbps trace at a campus edge.

B. *Unbiased estimators*

We now show how to construct unbiased estimators for packet and byte counts per flow. Unbiased estimators are critical mainly for volume estimation tasks without which errors can be really high, especially for aggregates constructed from individual flow records. We begin our discussion by providing an estimator of per-flow packet counts. Note that since host selection process is hash-based, no packets are lost before selecting a host; thus, the packet count estimate is dependent only on the p_f and p_p probabilities.

Estimation of per-flow packet counts. Let s be the actual number of packets for a flow f and c be the total number of packets sampled in the counter. The unbiased estimator $\widehat{s}$ for the number of packets is given by the following equation.

$$\widehat{s} = \frac{1}{p_f} + \frac{c-1}{p_p} \tag{1}$$

Proof: Intuitively, packet selection process for a given flow packet counter can be thought of as a sequence of rounds, with each round involving a sequence of unsampled packets finally terminating with a sampled packet. The final value of the counter c indicates the total number of rounds. Let us suppose $\widehat{s_i}$ be the random variable indicating the set of packets comprising the round i. Assuming packets are sampled with probability p_i within the ith round. The unbiased estimator $\widehat{s_i}$ is given by $\widehat{s_i} = 1/p_i$, which is the standard unbiased estimator for a geometric random variable.

As the packet sequence length is the sum of the lengths of individual rounds, $\widehat{s} = \sum_{k=1}^{c} \widehat{s_i}$. The probability for the first round $p_1 = p_f$, and for all other rounds, $p_j = p_p$, $1 < j \leq c$. Also, these individual random variables are independent of each other. Thus, the unbiased estimator for the packet count for a flow can be computed as follows.

$$\widehat{s} = \frac{1}{p_f} + \sum_{k=1}^{c-1} \frac{1}{p_p} = \frac{1}{p_f} + \frac{c-1}{p_p}$$

Variance estimate. The variance of this estimator can be computed similarly as follows. First we compute the variance of the individual s_is as follows.

$$Var[\widehat{s_i}] \quad = \quad \frac{1-p_i}{p_i^2} \tag{2}$$

The above expression is the variance estimate of a standard geometric random variable. Similar to the mean estimate, we can sum the individual variances to obtain the total variance of the estimate.

$$Var[\widehat{s}] = Var[\sum_{k=1}^{c} \widehat{s_i}] = \frac{1-p_f}{p_f^2} + (c-1)\frac{1-p_p}{p_p^2} \tag{3}$$

Estimation of per-flow byte counts. Let b_i $(1 \leq i < l)$ represent the byte size of individual packets for a given flow. The total byte count of a flow is represented by $b = \sum_{i=1}^{l} b_i$. Let S_c be the set of indices of sampled packets with probability, p_p, and c $(1 \leq c \leq l)$ denotes the cardinality of the set, $(|S_c|)$. Let b_{first} represent the size of the first packet sampled. An unbiased estimator $\widehat{b}$ of per-flow byte count b is as follows.

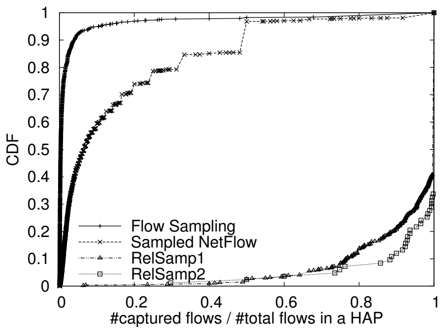
$$\widehat{B} = \frac{b_{first}}{p_f} + \sum_{i \in S_c} \frac{b_i}{p_p} \qquad (4)$$

We can prove that this estimator is unbiased in a similar fashion to the packet count estimator. Essentially, the byte count estimators for each of the s_i sequence of packets is given by $\widehat{B_i} = b_i/p_i$, where b_i is the byte count of the sampled packet in round i. We can compute the estimate for byte count $\widehat{B}$ trivially by adding up the individual $\widehat{B_i}$s. Variance of the byte count estimate, on the other hand, is hard to compute this way, unless we assume packet sizes are distributed uniformly. This assumption is not true in general; we have not yet been able to compute a general formula for estimating this and consider it part of future work.

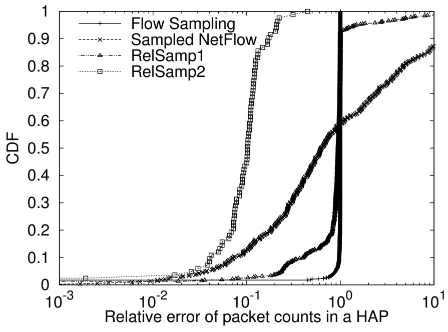
IV. Evaluation

We evaluate RelSamp in order to answer the following questions: 1) How effective is RelSamp in sampling related flows that belong to an application session? 2) Are the estimators in RelSamp unbiased? What is the relative inaccuracy in estimating flow volumes? 3) What are the limitations of using random packet sampling in enabling traffic classification? How effective is RelSamp in ensuring better classification accuracy as compared to random packet sampling?

We answer these questions with the help of a prototype of RelSamp we built by extending an open-source NetFlow called YAF [9]. While there exist several variants of NetFlow, we mainly compare with flow sampling and sampled NetFlow since none of the variants (*e.g.*, sketch-guided sampling [10]) are geared toward preserving the application structure similar to our goal. We evaluate the efficacy of RelSamp with the help of real traffic traces. The first dataset, *CAMPUS*, is an OC-192 (10 Gbps) packet-header trace collected at the edge of a large university campus. The trace is an hour long, and consists of about 140 million flows and 1,293 million packets. The purpose of using this trace is to answer questions about how to choose the parameters of RelSamp in edge network settings. Our second dataset, *DORM*, is a packet trace with full payloads collected from a router on a large dormitory building in the campus. Since we have no access to the packet payload from the CAMPUS trace, this full-payload trace allows us to evaluate the implications of RelSamp for traffic classification by enabling deep packet inspection (DPI) techniques to establish ground truth, i.e., determine the actual application (see Section V). The trace is about 13 hours long. We filter out traffic local to the university, and the resulting trace consists of around 214 million packets distributed over 8.5 million flows and carrying around 139 GBytes of volume.



(a) Flow coverage



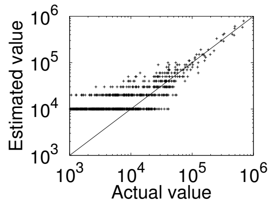
(b) HAP volume error

Fig. 4. Flow coverage and volume estimate of a HAP. We use $\tau = 30s$ and use the CAMPUS trace for this experiment.

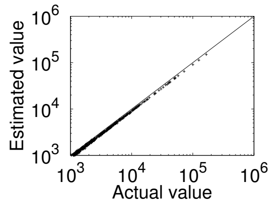
A. Sampling related flows

We explore the effectiveness of RelSamp in sampling flows corresponding to the same HAP, and the sensitivity of the results to the parameters of RelSamp using two configurations; for a setting called RelSamp1, we use $p_h = 0.03$, $p_f = 0.76$, and $p_p = 0.0001$; another setting, RelSamp2 has $p_h = 0.005$, $p_f = 1.0$, and $p_p = 0.3$. The two different sampling settings result in similar effective sampling rate of 0.001, but have different proportion of p_h and p_p. Higher p_h essentially chooses higher number of hosts for consideration, while lower p_p results in slightly higher error for individual flows. Depending on the importance of different objectives, one could choose different settings. For comparison, we use two other sampling schemes, Sampled NetFlow with 0.001 sampling rate and flow sampling (with 0.0015 flow sampling rate such that 0.001 fraction of packets are sampled).

Our primary evaluation metrics are *flow coverage*, which we define as the fraction of the flows captured by Sampled NetFlow or RelSamp per HAP, and *volume estimate* defined as the total number of packets per HAP captured by these algorithms. Figure 4(a) plots the CDF of the flow coverage obtained across the HAPs. Only HAPs with at least two flows are considered; considering HAPs with one flow will shift the curves slightly to the left. Each curve corresponds to results with a particular sampling algorithm (Flow sampling, NetFlow, and two settings of RelSamp). From the figure, we can make several observations. First, NetFlow (topmost curve) is the least effective in ensuring flow coverage—only 10% of the



(a) RelSamp1



(b) RelSamp2

Fig. 5. Scatter plot of actual and estimated packets counts.

HAPs see a flow coverage of 50% or more, with a median flow coverage of only 10%. Second, as we move from RelSamp1 to RelSamp2, the flow coverage curve moves to the right. We can see that RelSamp2 performs better than RelSamp1 due to the fact that p_f is higher in RelSamp2 setting than RelSamp1.

Our most aggressive setting RelSamp2 achieves more than 90% flow coverage for over 90% of flows. Our RelSamp1 and RelSamp2 curves clearly indicate almost **5-10x** increase in the median flow coverage compared to Sampled NetFlow and almost **10x** increase compared to the flow sampling curve, which performs the worst in all the experiments due to its inability to preserve any application semantics. As an aside, we note that with the RelSamp2 setting, even though $p_f = 1.0$, the flow coverage can be observed to be less than 1 in Figure 4(a). This is because there are multiple normal flows with the same flow key in a HAP while NetFlow's (in)active timeouts can split a single flow into several flow records. That makes it difficult to match the NetFlow's output with HAPs exactly. Thus, we treat all flows with the same flow key as a flow.

In Figure 4(b), we compare the relative error in estimating a HAP's volume for different sampling schemes, *i.e.*, we consider the error in estimating the total volume across all flows that constitute a HAP compared to the ground truth. We can observe once again that flow sampling curve is significantly worse than the rest, with a relative error of almost 100% for more than 95% of HAPs. Sampled NetFlow performs slightly better compared to flow sampling, but still, the median error is close to 100% error. RelSamp1 performs better than both flow sampling and Sampled NetFlow, but because the individual flow volume estimates are not that accurate, it also suffers from worse error. RelSamp2 performs the best since it is the most aligned with our goal, namely, higher flow coverage and accurate flow volume estimates ensured by the higher value of p_p, which is set at 30%.

One could argue that RelSamp2 is *strictly* better than RelSamp1, as it achieves a smaller relative error of HAP volume estimates, and captures more flows per HAP. While this is true, the advantage of RelSamp1 is that it uses a larger fraction of hosts. This advantage may matter when one considers other application characteristics where such a difference is important. Overall, our architecture provides the flexibility to choose arbitrary settings depending on the goals of a network operator.

While we only discussed two settings of parameters here, we briefly provide the intuition about which parameter controls what characteristic of traffic. The host selection probability p_h controls the total number of hosts, and thus the total number of packets, directly impacting the aggregate volume estimation accuracy. On the other hand, flow selection probability p_f governs application awareness, while packet selection probability p_p dictates the accuracy of per-flow statistics. Thus, there exists a clear trade-off in configuring these three parameters depending on the needs of the network operator, measurement location, link capacity, and router resource constraints. As application mix and characteristics evolve, operators need to tune these parameters; we will pursue techniques to automatically tune these parameters as part of our future work.

B. *Unbiasedness of estimators*

We empirically validate our unbiased estimators next. Figure 5(a) first shows the scatter plot of actual and estimated packet counts for flows containing at least 1,000 packets using the RelSamp1 setting. The two-sided errors from the figure, though not strictly conclusive, indicate the unbiased nature of our estimator. As flow size increases, the actual and estimated flow sizes converge and the relative error becomes smaller. We observe a similar trend for the scatter plot corresponding to the RelSamp2 setting as shown in Figure 5(b), except that the estimates are much more accurate than that of RelSamp1. The reason for this is quite obvious; the packet sampling probability p_p under the RelSamp2 setting is quite high (30%) compared to the (0.01%) setting in RelSamp1. This allows RelSamp2 to be more accurate in the individual flow volume estimates.

V. Impact on traffic classification

Network operators need to classify traffic to enable services such as traffic differentiation and estimating volumes of individual applications. While researchers have proposed several approaches based on host-behavior [4], [7] and machine-learning techniques [3], [5] for classification, most of this work assumes unsampled data. Our focus, in contrast, is to study the impact of sampling on traffic classification.

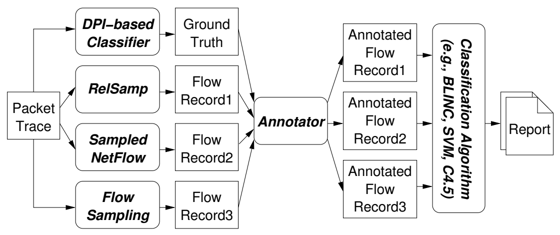
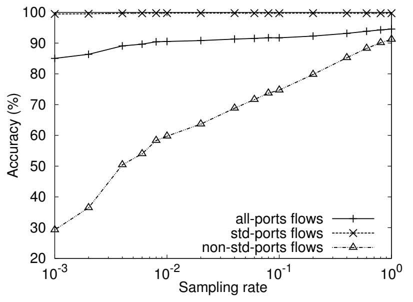
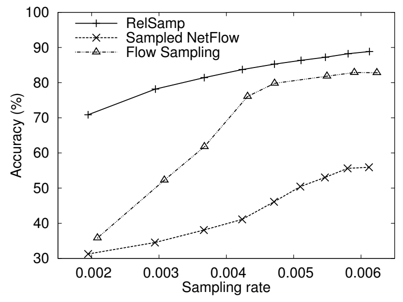


Fig. 6. Dataflow used for evaluation of classification techniques.

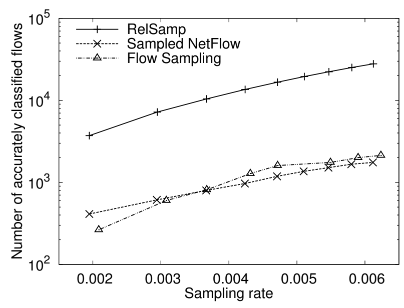
We shed light on (i) the effect of sampling on traffic classification techniques; and (ii) the effectiveness of RelSamp in aiding traffic classification when compared to NetFlow and Flow Sampling [8]. In our study, we analyze BLINC, a host-behavior based classifier, and two machine-learning algorithms: C4.5 and Support Vector Machine (SVM). We start our discussion by describing the methodology used to evaluate them, and end with detailed results.



(a) Accuracy for different categories of flows generated by Sampled NetFlow



(b) Accuracy for *non-std-ports flows*



(c) Number of accurately classified flows for *non-std-ports flows*

Fig. 7. BLINC's classification accuracy results.

A. Methodology

Our evaluations are conducted using packet traces collected with full packet payload required to establish the ground-truth. Figure 6 illustrates the dataflow to evaluate classification techniques. Four different methodologies are parallelly applied to the packet-level trace: a DPI-based classifier (to obtain ground truth), RelSamp, Sampled NetFlow, and Flow Sampling. Raw unsampled flows are annotated with application types using a deep packet inspection (DPI) tool called `tstat` [11]. `tstat` provides ground truth by annotating each flow with its application type after parsing its content searching for signatures of applications it can recognize. In our evaluations, we ignore flows for which `tstat` is unable to determine their application type (*e.g.*, encrypted flows). In our trace, less than 8% of the total flows were unable to be classified by `tstat`. The flows generated by RelSamp, Sampled NetFlow and Flow Sampling are annotated by correlating them with the unsampled flows. Sampled flows are then fed to the different classification techniques.

Traffic flows conforming to well-known standard ports are easy to classify using a simple classifier that takes port number information for classification. However, classification of flows that do not use their well-known standard ports is far more challenging. We therefore dissect traffic into three categories depending on their use of standard ports: (i) *std-ports flows* which are flows that use any of the standard well-known ports for that flow type (*e.g.*, port 80 for Web, 110 for POP3, and 443 for SSL); (ii) *non-std-ports flows* which are flows that do not use the standard known port for the flow's application type; and (iii) *all-ports flows* which include all flows independent of their ports. In the DORM trace, we found that 42% of the total flows were *std-ports flows* and the remaining 58% of the flows were of the *non-std-ports flows* type. In the rest of the analysis, we mainly focus on the second category.

We employ *Reverse BLINC* [12], an extension of original BLINC, with the default values of configuration parameters. Reverse BLINC overcomes the limitation of misclassification of non-bidirectional traffic which existed in the original BLINC version. For machine learning classification, we use a well-known data mining software suite called WEKA [13]. We use Sequential Minimal Optimization (SMO) [14] for training SVM, and used parameter settings and features that have been shown to work well in prior work [12].

We test classification techniques using one hour of packet data from the DORM trace. The same trace is used as input for all classification techniques. Supervised machine learning techniques also require training, so we use another hour from the same trace to train both SVM and C4.5. For each sampling algorithm and sampling rate, we train and test SVM and C4.5 with data sampled using those settings. For example, if we are to test SVM on traffic sampled with rate 0.001, the same sampling rate of 0.001 is also used for training. We take this approach to use features consistently during training and testing, which in turn, can ensure accuracy of the classifiers.

For the parameter setting of RelSamp, while we fix $p_h = 0.2$ and $p_p = 0.0001$, we varied p_f from 0.1 to 0.9. We set parameter values which are higher than RelSamp1 setting because host population and flow sizes in DORM trace are smaller than those in CAMPUS trace. In addition, because we wish to investigate the influence of p_f on traffic classification, we do not fix any given p_e, but let it vary freely by varying p_f values. Thus, p_e is higher than 1 in 1024 typically used in OC-192 link, and ranged between 1 in 200 to 600.

B. Results

Figure 7(a) studies the impact of sampling on classification accuracy with BLINC on the DORM trace. We define accuracy as c/t, where c is the total number of correctly classified flows, and t is the total number of flows. The x-axis shows the sampling rate used and the y-axis shows the accuracy of BLINC for the three categories of flows mentioned earlier. The overall accuracy for all flows reduced from around 95% down to 85% as the sampling rate was reduced from 1 to 0.001, which might mislead one to conclude that sampling does not affect classification accuracy. The overall accuracy includes both *std-ports flows* as well as *non-std-ports flows*, on which BLINC performs differently. When only *std-ports flows* were considered, the accuracy was almost 100% across the range of sampling rates as we expected. However, when only *non-std-ports flows* were considered—arguably, the regime where the need for sophisticated classifiers is most critical—the accuracy decreased significantly from 90% over unsampled data all the way down to 30% with a 1 in 1000 sampling.

Figure 7(b) compares the accuracy of BLINC when RelSamp is used compared to when Sampled NetFlow and Flow

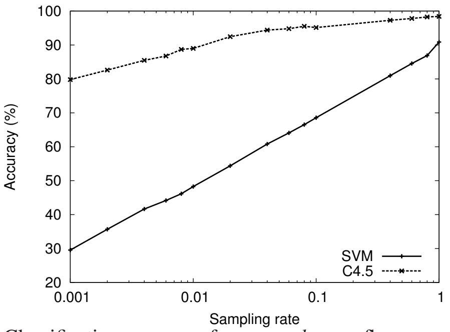
Sampling are used, focusing mainly on *non-std-ports flows*. To ensure a fair comparison, we require that the packets are sampled at the same effective rate with both schemes. Compared to Sampled Netflow, the benefits of using RelSamp are significant for all sampling rates considered—for instance, for an effective sampling rate of 0.002, the accuracy is 70% with RelSamp and only 30% for Sampled NetFlow.

While RelSamp outperforms Flow Sampling from 0.002 to 0.004, the accuracy gap between RelSamp and Flow Sampling only becomes about 8% after that range. To understand this result better, we investigated how much accuracy Flow Sampling achieved for *std-ports flows*. Interestingly, the classification results were worse than those of RelSamp and Sampled NetFlow (not shown for the brevity). Both of them approximately achieve over 97% accuracy, but Flow Sampling only achieves about 80% accuracy. Further, we checked total number of *std-ports flows* as well as *non-std-ports flows* for Sampled NetFlow and Flow Sampling. It turned out that while Sampled NetFlow over-samples *std-ports flows* rather than *non-std-ports flows*, the ratio between two is similar in case of Flow Sampling. This is because Flow Sampling is unbiased in flow size but there exists biasness in flow size between two categories in our trace. Note that classification in BLINC is triggered by a threshold in number of flows constituting a graphlet (a signature to describe host's common behavior for a particular application). Thus, while the accuracies of both categories of flows were influenced in case of Flow Sampling because it balances the number of sampled flows in both categories, Sampled NetFlow had worse accuracy in *non-std-ports flows* classification because it less samples those flows.

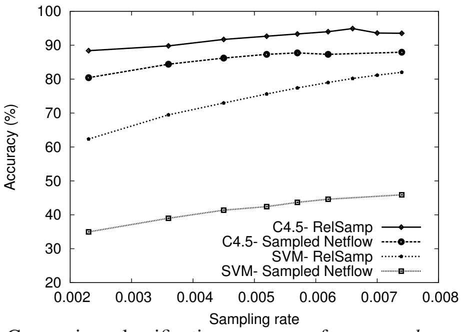
We also looked at the actual number of correctly classified flows for each sampling schemes. At the right-most data point in Figure 7(c), while RelSamp classifies about 28,000 flows correctly, other two methods successfully classify only about 2,000 flows. On the whole, RelSamp accurately classified roughly **10 times** more number of flows than Sampled NetFlow and Flow Sampling. Therefore, our RelSamp outperforms the other two sampling methods in terms of accuracy as well as the absolute number of correctly classified flows.

We now discuss the reasons for the significantly improved accuracy with RelSamp. Host-behavior based classifiers such as BLINC work by observing communication patterns of hosts. Sampling affects BLINC by providing distorted hosts' profiles that do not reflect the actual social communication between them. Actually, we found an example from our dataset for the p2p graphlet to show distorted hosts' profiles. RelSamp was able to capture 55 flows which exceeds BLINC's threshold (*fanout* cardinality) for the p2p graphlet, but sampled Netflow captured only 3 flows which is significantly below the threshold leading to BLINC classifying them as *unknown*.

Figure 8(a) studies the impact of sampling by Sampled Netflow on C4.5 and SVM for *non-std-ports flows*. As sampling rate was lowered, accuracy decreased from 90% down to 30% for SVM, and 98% to 80% for C4.5. We believe the degradation with SVM occurs since it has the problem of training $overfitting$ [15], and thus requires the non-trivial process of choosing a representative training dataset (*e.g.*, using *data-categorization* mechanisms) to produce better maximum-margin hyperplanes [16]. In contrast, C4.5 appears relatively more robust to sampling. We hypothesize this may be due to the different underlying machine learning algorithm used, with the fine grained nature of the decision tree approach potentially leading to more robustness. A more extensive study using a wide range of traces is required to draw definite conclusions regarding the relative robustness of different classification algorithms, and is deferred to part of future work.



(a) Classification accuracy for *non-std-ports* flows generated by Sampled NetFlow



(b) Comparing classification accuracy for *non-std-ports flows* generated by Sampled NetFlow and RelSamp

Fig. 8. C4.5 and SVM classification results.

Figure 8(b) compares the classification accuracy of C4.5 and SVM for *non-std-ports flows* obtained with Sampled NetFlow and RelSamp. The figure shows that RelSamp outperformed Netflow by around 10% in C4.5 and 35% with SVM. We believe this is because RelSamp captures more flows than Sampled NetFlow (even though the number of packets sampled in each scheme is the same), which potentially provides better data for the training phase.

VI. Related Work

Due to the importance of measurements in several network management tasks, there exists a lot of prior work [17], [18], [19], [20], [21] in architecting better sampling-based passive measurement solutions. Despite their existence, to the best of our knowledge, we are the first to introduce this notion of related sampling that can be exploited to make sampled flow records preserve application structure. In this section, we outline some of these solutions and discuss how they differ from our RelSamp architecture.

Several researchers have observed glaring deficiencies in Sampled NetFlow and proposed solutions to address some of them. For instance, Kompella *et al.* propose Flow Slices as a solution to allow better tuning knobs for controlling memory and CPU utilization in [18]; the notion of related sampling is, however, absent in Flow Slices. Sekar *et al.* have proposed cSamp [22] for network-wide flow monitoring with a goal to minimize redundancies in routers sampling packets independently. Traffic Sentinel [23] is another approach similar in spirit to cSamp. RelSamp, on the other hand, is designed to operate within a router; we will pursue extending RelSamp to a coordinated setting as part of our future work. Beyond sampling frameworks, a few past works have focused on providing network operators with complete flexibility in choosing which flows they want to sample. For example, Yuan *et al.* devised ProgME [19] that provides operators the flexibility to configure hyper-spaces called flowsets. While it offers operators with great flexibility, it is not clear how to exactly define these flowsets to preserve application structure. One other related recent effort is FlexSample [20] by Ramachandran *et al.*, in which they provide a simple language to specify groups of flows of interest using tuples on specific packet header fields. It requires maintaining extra state for approximate checking of the predicates and is designed to provide sophisticated monitoring of traffic subpopulations. RelSamp, on the other hand, does not require any extra state; it is designed to be simple for network operators to implement related sampling mainly geared toward application monitoring. Influence of sampled traffic on anomaly detections has been studied in the past [24]. Our work, although not directly related, may benefit anomaly detection as well.

Traffic classification is one of the main applications of RelSamp which we have studied in this paper. In general three main approaches emerged—deep packet inspection (DPI) [11], [25], behavior-based [4], [7], and flow-feature based [3], [5], [6]. While in high speed networks the use of packet sampling is inevitable, all previous works have mainly focused on unsampled traffic data. Two of these works [6], [5] have hypothesized that the accuracy of their method will degrade fairly quickly under packet sampling, but neither investigates it further. Our work, in contrast, provides no new classification mechanism, but instead provides a sampling scheme that can improve the accuracy of traditional classification techniques.

VII. Conclusion

While the wide availability of NetFlow across many modern routers makes it an ideal candidate for continuous, low-cost monitoring of network application traffic at enterprise edge routers, the sampling algorithms employed by NetFlow today are inadequate to capture application behavior. In this paper, we have presented RelSamp, an architecture based on the key idea that related flows, part of the same application session, are sampled with higher probability. Our evaluations on real traces show the importance and viability of a related sampling approach. We demonstrate that RelSamp is capable of obtaining 5-10x more flows per application session compared to Sampled NetFlow and increases the classification accuracy of BLINC, SVM and C4.5 up to 50% in comparison with the flows output by Sampled NetFlow.

Acknowledgments

We thank William Harshbarger, Greg Hedrick at Purdue ITaP for their immense help in obtaining the network traces. This work was supported in part by NSF Award CNS 0831647, 1017898, 1017915 and a grant from Cisco Systems.

References

[1] “Endace,” http://www.endace.com.
[2] T. Karagiannis, A. Broido, N. Brownlee, K. Claffy, and M. Faloutsos, “Is P2P dying or just hiding?” in *IEEE Globecom, 2004*.
[3] A. Moore and D. Zuev, “Internet traffic classification using bayesian analysis techniques,” *ACM SIGMETRICS Performance Evaluation Review*, vol. 33, no. 1, pp. 50–60, 2005.
[4] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, “BLINC: multilevel traffic classification in the dark,” in *ACM SIGCOMM, 2005*.
[5] S. Zander, T. Nguyen, and G. Armitage, “Automated traffic classification and application identification using machine learning,” in *Conference on Local Computer Networks*, 2005.
[6] L. Bernaille, R. Teixeira, and K. Salamatian, “Early application identification,” in *ACM CoNEXT 2006*, 2006.
[7] M. Iliofotou, P. Pappu, M. Faloutsos, M. Mitzenmacher, S. Singh, and G. Varghese, “Network monitoring using traffic dispersion graphs (tdgs),” in *IMC, 2007*.
[8] N. Hohn and D. Veitch, “Inverting sampled traffic,” *IEEE/ACM Transaction on Networking*, vol. 14, no. 1, pp. 68–80, 2006.
[9] “YAF: Yet Another Flowmeter,” http://tools.netsa.cert.org/yaf/.
[10] A. Kumar and J. Xu, “Sketch guided sampling - using on-line estimates of flow size for adaptive data collection,” in *INFOCOM*, 2006.
[11] M. Mellia, R. Lo Cigno, and F. Neri, “Measuring IP and TCP behavior on edge nodes with Tstat,” *Computer Networks*, vol. 47, no. 1, pp. 1–21, 2005.
[12] H.-C. Kim, K. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee, “Internet traffic classification demystified: Myths, caveats, and the best practices,” in *ACM CoNEXT, 2008*.
[13] “WEKA: Data-mining Software in Java.” http://www.cs.waikato.ac.nz/ml/weka.
[14] J. Platt, “Sequential minimal optimization: A fast algorithm for training support vector machines,” *Advances in Kernel Methods-Support Vector Learning*, 1999.
[15] I. Mierswa, “Controlling overfitting with multi-objective support vector machines,” in *GECCO, 2007*.
[16] T. Hiroyasu, M. Nishioka, M. Miki, and H. Yokouchi, “Application of MOGA Search Strategy to SVM Training Data Selection,” in *Proceedings of the 5th International Conference on Evolutionary Multi-Criterion Optimization*. Springer, 2009, pp. 125–139.
[17] C. Estan, K. Keys, D. Moore, and G. Varghese, “Building a better netflow,” in *ACM SIGCOMM*, 2004.
[18] R. R. Kompella and C. Estan, “The power of slicing in internet flow measurement,” in *IMC*, 2005.
[19] L. Yuan, C.-N. Chuah, and P. Mohapatra, “ProgME: towards programmable network measurement,” in *ACM SIGCOMM*, 2007.
[20] A. Ramachandran, S. Seetharaman, N. Feamster, and V. Vazirani, “Fast monitoring of traffic subpopulations,” in *ACM/USENIX IMC*, 2008.
[21] M. Saxena and R. R. Kompella, “A framework for efficient class-based sampling,” in *INFOCOM*, 2009.
[22] V. Sekar, M. K. Reiter, W. Willinger, H. Zhang, R. R. Kompella, and D. G. Andersen, “cSAMP: a system for network-wide flow monitoring,” in *USENIX NSDI*, 2008.
[23] “Traffic Sentinel,” http://www.inmon.com/products/trafficsentinel.php.
[24] J. Mai, C.-N. Chuah, A. Sridharan, T. Ye, and H. Zang, “Is sampled data sufficient for anomaly detection?” in *ACM/USENIX IMC*, 2006.
[25] T. Choi, C. Kim, S. Yoon, J. Park, B. Lee, H. Kim, H. Chung, and T. Jeong, “Content-aware Internet application traffic measurement and analysis,” in *IEEE/IFIP NOMS 2004*.