Self-Learning Classifier for Internet traffic

Self-Learning Classifier for Internet Traffic

Luigi Grimaudo*, Marco Mellia*, Elena Baralis*, Ram Keralapura+
* Politecnico di Torino, Italy. + Narus Inc, Sunnyvale, California.

***Abstract*—Network visibility is a critical part of traffic engineering, network management, and security. Recently, unsupervised algorithms have been envisioned as a viable alternative to automatically identify classes of traffic. However, the accuracy achieved so far does not allow to use them for traffic classification in practical scenario.**

In this paper, we propose SeLeCT, a Self-Learning Classifier for Internet traffic. It uses unsupervised algorithms along with an adaptive learning approach to automatically let classes of traffic emerge, being identified and (easily) labeled. SeLeCT automatically groups flows into pure (or homogeneous) clusters using alternating simple clustering and filtering phases to remove outliers. SeLeCT uses an adaptive learning approach to boost its ability to spot new protocols and applications. Finally, SeLeCT also simplifies label assignment (which is still based on some manual intervention) so that proper class labels can be easily discovered.

We evaluate the performance of SeLeCT using traffic traces collected in different years from various ISPs located in 3 different continents. Our experiments show that SeLeCT achieves overall accuracy close to 98%. Unlike state-of-art classifiers, the biggest advantage of SeLeCT is its ability to help discovering new protocols and applications in an almost automated fashion.

I. Introduction and motivation

A critical part of network management and traffic engineering is the ability to identify applications and protocols originating traffic flows. To provide network visibility, in the last years several classification techniques have been proposed (see [1], [2] and references therein). *Deep packet inspection* (DPI) [1], and *behavioral* techniques have been investigated since the seminal work of [3].

Both approaches share key limitations. First, they require *training*, i.e., the availability of a signature, or of a training set. Second, and most critical, the classifiers *can identify only the specific applications they have been trained for* and they cannot identify new applications, or changes in the applications' protocol or behavior, unless an expensive re-training phase is entered. Designing a classification engine capable of automatically identifying new emerging protocols is still an open and challenging research topic.

In this paper, we propose SeLeCT, a novel algorithm that overcomes the limitations highlighted above. We leverage unsupervised data mining algorithms to automatically split traffic into homogeneous subsets (or clusters) by using simple layer-4 metrics, like segment size and inter-arrival time. These features are known to carry valuable information for traffic classification [2]. However, they are known *not* to perform well in the context of unsupervised (i.e. clustering) algorithms [4]. Hence we have to adopt some ingenuity to improve cluster homogeneity. To overcome the limitation of previous proposals, we design an *iterative clustering* procedure in which a *filtering phase* follows each clustering phase to eliminate possible outliers. The filtering phase is based on the information provided by the server port number which instead turns out to be hard to include in clustering algorithms.

The research leading to these results has received funding from the European Union under the FP7 Grant Agreement n. 318627 (Integrated Project "mPlane")

TABLE I
Datasets used for performance evaluation.

name	DateTime	Place	Type	IP	Flow
DS-1	Aug05 1pm	S.America	backbone	108k	527k
DS-2	Sep10 10am	Asia	backbone	111k	1.8M
DS-3	Aug11 2am	Europe	access	111k	885k
DS-4	Aug11 5pm	Europe	access	190k	2.3M

Using traffic traces collected in different years from various ISPs located in 3 different continents, we show that SeLeCT generate clusters with excellent properties: few, and very pure clusters. This allows to quickly inspect each cluster, allowing an easy manual labeling of flows.

Once labels are assigned, SeLeCT will automatically inherit them for classification of future batches. We refer to this as *adaptive* or *progressive* learning since flows labeled in the past are used to seed the subsequent datasets. Notably, this will minimize the bootstrapping effort required to label applications, and manual intervention is required only for the initial label assignment.

In the data mining community, the semi-supervised learning approaches were proposed first [5], [6], where labeled data are mixed to new samples before clustering. Labeled data is thus exploited to assign labels to unlabeled data. Our labeling process is similar to [5]. Due to its iterative refinement process, SeLeCT is particularly suited to model Internet traffic changes, because it allows a seamless adaptation of the obtained traffic classes to traffic pattern evolution. In the context of traffic analysis, [7] is one of the first work that proposes clustering techniques to obtain insights about the traffic. Then, several work apply unsupervised methodologies to traffic analysis problem [4], [8], [9], [10], [11]. However, moderate accuracy ($\leq 80-85\%$) has been achieved so far. Furthermore, few traffic classes have been considered (e.g., P2P, HTTP, Email, FTP). For example, in [8] authors leverage the standard k-means to construct clusters. Then, a semi-supervised algorithm is proposed where a simple voting scheme is used to extend the dominant label to the whole cluster. Performance when only coarse classes are considered shows that, to achieve accuracy up to 85%, a large number of clusters must be used (≥ 400). SeLeCT follows similar principles, extending the idea with i)

iterative filtering and ii) multi-batch seeding. Both significantly boost overall performance (accuracy larger than 98%), while the number of clusters is reduced to less than 150. A similar idea of an unsupervised algorithm is described in [9]. The authors leverage the entropy of the packet payload to find out clusters of traffic, focusing specifically on UDP traffic.

The excellent properties of SeLeCT helped in unveiling the presence of unknown/undesired classes too (e.g., Apple push notification, Bot/Trojan, or Skype authentication traffic), i.e., offering almost automatic and very fine-grained traffic visibility.

II. Problem statement and Datasets

We consider *directed traffic flows* as the objects to classify. A directed flow, or flow for short, is defined as the group of packets that have the same five tuple $F = \{srcIP, dstIP, srcPort, dstPort, protocol\}$. Note that packets going in opposite directions belong to two directed flows. This allows the classifier to work even in presence of asymmetric routing (backbone networks for instance).

We assume all packets traversing a link are exposed to the classifier which keeps track of per-flow state. For each flow F, a set of features is collected. The goal of SeLeCT is to assign a proper application to each flow based on the sole knowledge of the flow feature. In this work, we choose behavioral features that are well known to carry useful information about the application and protocol used at the application layer [1], [2]. In particular, we select: (i) The server port $srvPrt$, (ii) the length of the first n segments with payload, and (iii) their corresponding inter-arrival-time[1].

Table I summarizes the main characteristics of the datasets (DS) we use for performance evaluation. We collected four different traces from access and backbone networks of large ISPs. Each dataset is a 1-hour long complete packet trace including the packet payloads. We selected these traces to create a very heterogeneous benchmark set.

For each trace, we generate two separate datasets - the set of flows originated by clients (i.e., hosts actively opening the TCP connection) and the set of flows originated by servers (i.e., hosts that replied to the connection request). A letter 'C' (client-to-server) or 'S' (server-to-client) is appended at the dataset name when needed.

As mentioned before, only flows that have at least n data packets are considered by SeLeCT. We checked the fraction of traffic that SeLeCT targets. As expected, the large majority of the flows are "mice", i.e., flows with few packets. For instance, 90% of client flows have no more than 6 data packets. However, they account for no more than 1% of the volume of traffic. Thus, by considering flows that have at least 6 data packets: (i) we allow a richer description of each flow characteristics (ii) we are discarding the large majority of mice flows and (iii) we are looking at more than 99% of traffic volume. Based on these observations, in the rest of the paper we use $n = 6$.

[1]The choice of which feature to use is a matter of optimization. Given the already excellent performance of SeLeCT, we leave this for future work.

1: **Main**()
2: *Output*: set $\mathcal{C}$ of labeled clusters
3: $\mathcal{S} = \emptyset$
4: **while** (newbatch $\mathcal{B}$) **do**
5: **ProcessBatch**($\mathcal{B}$, $\mathcal{S}$, $\mathcal{C}$, $\mathcal{NS}$)
6: $\mathcal{S} = \mathcal{NS}$
7: **end while**
8: **ProcessBatch**($\mathcal{B}$, $\mathcal{S}$, $\mathcal{C}$, $\mathcal{NS}$):
9: *Input*: Set $\mathcal{B}$ of new flows, set $\mathcal{S}$ of seeds
10: *Output*: set $\mathcal{C}$ of labeled clusters, set $\mathcal{NS}$ of new seeds
11: $\mathcal{B}'$ = $\mathcal{B} \cup \mathcal{S}$ {Merge new flow and seeding set}
12: $\mathcal{C}'$ = doIterativeClustering($\mathcal{B}'$);
13: $\mathcal{C}$ = doLabeling($\mathcal{C}'$);
14: $\mathcal{NS}$ = extractSeeds($\mathcal{C}$);

Fig. 1. SeLeCT Main loop algorithm.

III. The SeLeCT algorithm

We consider a scenario in which traffic is sniffed in real time and new flows enter the system continuously. Flows are processed in *batches*. SeLeCT analyzes each batch of newly collected flows via the **ProcessBatch()** in Fig. 1. This function takes in input $\mathcal{B}$, the set of new flows, and $\mathcal{S}$, the set of *seeding flows*, i.e., flows already analyzed in past batches for which SeLeCT was able to provide a label. Its main steps (see Fig. 1) are (i) clustering batch data to get homogeneous subsets of flows (function **doIterativeClustering()**), (ii) flow label assignment (function **doLabeling()**), and (iii) extraction of a new set of seeds (function **extractSeeds()**). In the following we detail each step of the batch processing.

A. Iterative clustering: it is the core of SeLeCT. It exploits the k-means clustering algorithm [12] to group flows into subsets or *clusters* which are possibly generated by the same applications. In this context, it is natural to consider two flows with similar packet length and inter-arrival time to be close (i.e., to be likely generated by the same application). However, the same property does not hold for the $srvPort$ feature. For instance, two flows directed to port 25 and to port 80 are not more likely to be similar than two flows directed to port 80 and to port 62000. The $srvPort$ feature is a nominal feature [12], thus it cannot be included in Euclidean distance computations.

Still, the $srvPort$ is an important feature for traffic classification [2]. Two cases can be distinguished: protocols and applications i) running on one (or more) specific $srvPort$ on servers, or ii) running on a random $srvPort$ selected by each server. We denote them as *dominatedPort* and *randomPort* protocols respectively. In both cases, the $srvPort$ carries valuable information if applied as a *filter*.

We engineer an iterative procedure to identify clusters of flows in which the $srvPort$ information is used to *filter* elements in each cluster. We devise an iterative process, in which clustering and filtering phases alternate (see Fig. 3).

1) The filtering procedure: (Fig. 2) Filtering is performed on a single cluster at a time, the input cluster $\mathcal{I}$. First, **doFiltering()** discards clusters which have less than $minPoints$ flows to avoid dealing with excessively small clusters. Flows in these clusters are returned in set $\mathcal{U}$, the set of unclustered flows (lines 5-7).

The core activity of the filtering procedure is the identifica-

1: **doFiltering**($\mathcal{I}, \mathcal{C}, \mathcal{U}, \mathcal{DP}, portFraction, DominatingPhase$)
2: *Input*: cluster I of flows, $DominatingPhase$ flag
3: *Output*: set $\mathcal{C}$ of clusters, set $\mathcal{U}$ of noise, set $\mathcal{DP}$ of dominant ports
4: $\mathcal{DP} = \emptyset$
5: **if** $||\mathcal{I}|| < minPoints$ **then**
6: $\mathcal{U} = \mathcal{U} \cup \mathcal{I}$; return
7: **end if**
8: **if** $DominatingPhase$ **then**
9: {Processing dominatedPort cluster}
10: **if** (topPortFreq($\mathcal{I}$) $> portFraction$) **then**
11: $\mathcal{C}' =$ getFlows($\mathcal{I}$,dp)
12: $\mathcal{C} = \mathcal{C} \cup \mathcal{C}'$ {Add the filtered cluster to $\mathcal{C}$}
13: $\mathcal{R} = \mathcal{I} \setminus \mathcal{C}'$
14: $\mathcal{U} = \mathcal{U} \cup \mathcal{R}$ {Put discarded flows in $\mathcal{U}$}
15: $dp =$ dominantPort($\mathcal{I}$)
16: $\mathcal{DP} = \mathcal{DP} \cup \{dp\}$ {Record dominant port}
17: **else**
18: $\mathcal{U} = \mathcal{U} \cup \mathcal{I}$ {$\mathcal{I}$ flows must be reclustered}
19: **end if**
20: **else**
21: $\mathcal{C} = \mathcal{C} \cup \mathcal{I}$ {$\mathcal{I}$ is a good cluster at last}
22: **end if**

Fig. 2. Cluster filtering algorithm.

tion of *dominatedPort* clusters ($DominatingPhase$ is true). To this aim, the $srvPort$ distribution is checked. If the fraction of flows with the most frequent $srvPort$ in $\mathcal{I}$ exceeds the threshold $portFraction$, the cluster is a *dominatedPort* cluster. The flows involving the dominant $srvPort$ are clustered together and added to the set $\mathcal{C}$ of final clusters (line 11-12), while flows not involving the dominant $srvPort$ are removed and put in $\mathcal{U}$ (lines 13-14). The dominant port dp is included in the set $\mathcal{DP}$ of dominant ports (lines 15-16). If there is no dominant port, all flows from $\mathcal{I}$ are put in $\mathcal{U}$ (lines 17-18).

When $DominatingPhase$ is false, *randomPort* clusters are handled. In this case, cluster $\mathcal{I}$ (with all its flows) is simply added to the set of final clusters (line 21).

2) The iterative clustering procedure: (Fig. 3) It first iteratively generates dominated port clusters, and finally generates random port clusters. More specifically, the set of flows to be clustered is processed for $itermax$ iterations. At each iteration the set $\mathcal{U}$ of flows that are not yet assigned to any cluster is processed (lines 5-12). k clusters are formed using the well-known k-means algorithm [12] and assigned to $\mathcal{C}'$. Each cluster in $\mathcal{C}'$ undergoes a filtering phase (lines 8-11), which is looking for *dominatedPort* clusters only. The **doFiltering()** procedure returns in $\mathcal{U}$ flows that do not pass the filter and must be processed at the next iteration.

After $itermax$ iterations, *randomPort* clusters are handled. In this case, the information carried by the dominant port has been already exploited in previous phases, and the set of dominant ports $\mathcal{DP}$ contains the $srvPort$ that appeared as dominant in the past. Intuitively, if a $srvPort$ emerged as dominant port, then flows that have not been already put into $srvPort$ dominated clusters should be considered outliers. Hence, all flows involving any port in $\mathcal{DP}$ are removed from $\mathcal{U}$ (lines 13-15) before the final clustering and filtering phases (line 16-20) are completed.

B. Labeling: Once flows have been clustered, the

1: **doIterativeClustering**($\mathcal{B}$)
2: *Input*: Set $\mathcal{B}$ of flows to be clustered
3: *Output*: set of clusters $\mathcal{C}$
4: $\mathcal{U} = \mathcal{B}$, $\mathcal{DP} = \emptyset$
5: **for** ($step$=1; $step \leq itermax$; $step$++) **do**
6: $\mathcal{C}' =$ k-means($\mathcal{U}$)
7: $\mathcal{U} = \emptyset$, update($portFraction$)
8: **for** $\mathcal{I}$ in $\mathcal{C}'$ **do**
9: {look for *dominatedPort* clusters first}
10: doFiltering($\mathcal{I}$,$\mathcal{C}$,$\mathcal{U}$,$\mathcal{DP}$,$portFraction$,true)
11: **end for**
12: **end for**
13: **for** dp in $\mathcal{DP}$ **do**
14: delFlows($\mathcal{U}$,dp) {Discard flows still to $\mathcal{DP}$}
15: **end for**
16: $\mathcal{C}' =$ k-means($\mathcal{U}$)
17: **for** $\mathcal{I}$ in $\mathcal{C}'$ **do**
18: {look for *randomPort* clusters now}
19: doFiltering($\mathcal{I}$,$\mathcal{C}$,$\mathcal{U}$,$\mathcal{DP}$,0,false)
20: **end for**
21: return $\mathcal{C}$

Fig. 3. Iterative clustering algorithm.

doLabeling($\mathcal{C}'$) procedure assigns a label to each cluster. For each cluster $\mathcal{I}$ in $\mathcal{C}'$, flows are checked. If $\mathcal{I}$ contains some *seeding flows*, i.e., flows (belonging to $\mathcal{S}$) that already have a label, a simple voting scheme as in [8] is adopted: the label with the largest frequency will be extended to all flows in $\mathcal{I}$, possibly over-ruling a previous label for other seeding flows. More complicated voting schemes may be adopted (e.g., by requiring that the most frequent label wins by 50% or more). However, performance evaluation shows that the homogeneity of clusters produced by the iterative clustering procedure is so high that simple schemes work very nicely in practice.

1) Bootstrapping the labeling process: If no seeding flows are present, $\mathcal{I}$ is labeled as "unknown" and passed to the system administrator that should manually label the cluster. This will clearly happen during the bootstrapping of SeLeCT, when no labeled flows are present.

To address this issue, several solutions can be envisioned. For example, labels can be manually assigned by using the domain knowledge of the system administrator, supported by all the available information on the flows in the cluster (e.g., port number, server IP addresses or even the flow payload, if available). We show how easily this can be done in Sec. V. A second option is to use a bootstrapping flow set from some active experiments in which traffic of a targeted application is generated. Similarly, a set of bootstrapping flows can be generated by providing labels obtained by some other available traffic classification tools, e.g., any DPI tool.

In all cases, the complexity of the labeling process is reduced to the analysis of few clusters, instead of hundred of thousands of flows. Notice that this process is particularly easy for port dominated clusters, for which the port number is expected to carry a lot of valuable information. We illustrate this in Sec. V. This mechanisms can be also automated as suggested by [13], but this is outside the scope of this paper.

C - Self-seeding: Once some clusters have been labeled, SeLeCT is able to automatically reuse this information to process next batches. This is simply achieved by extracting

some *seeding flows* from labeled clusters by means of the **extractSeeds($\mathcal{C}$)** procedure. To avoid the bias due to classes having much more flows than others (unbalanced classes), we support the adoption of a *proportional sampling technique*. Let $numSeeds$ be the target number of seeding flows, i.e., $numSeeds = ||\mathcal{NS}||$. For each labeled cluster $\mathcal{I}$, a number of labeled flows proportional to the cluster size is extracted at random. That is $||\mathcal{I}||\frac{||\mathcal{NS}||}{||\mathcal{C}||}$ flows are randomly selected. This simple sampling process guarantees that all clusters will have a number of representatives in $\mathcal{NS}$ that is proportional to the cluster size. This mechanism enforces a self training process that allows the system to grow the set of labeled data and thus augment the coverage of the classification process.

IV. Experimental results

We use two separate advanced DPI classifiers to label flows and use these labels as our ground truth: NarusInsight[2] professional tool, and Tstat [14]. A total of 23 different protocols are identified including web (HTTP/S, RTSP, TLS), mail (SMTP/S, POP3/S, IMAP/S), chat (XMPP, MSN, YAHOOIM), peer-to-peer (BitTorrent, eMule, Gnutella, Fasttrack, Ares) and other protocols (SMB, FTP, Telnet, IRC). Flows that do not match any of the DPI rules or that DPIs label differently are labeled as "unknown". Each dataset has a different share of application labels, with a typical bias toward most popular protocols like HTTP and/or P2P that dominate the datasets; we do not report these details for the sake of brevity. In order to evaluate classification performance, we use standard metrics such as *overall accuracy, recall,* and *precision*.

Extensive parameter sensitivity analysis has been carried out and is provided in [15]. In the rest of this section, we report the main results considering the following parameter settings: Batch size $||\mathcal{B}|| = 10,000$, number of flows used for seeding $numSeeds = 8,000$, $minPoints = 20$, $itermax = 3$, $portFraction = 0.5$ for $step < itermax$, and $portFraction = 0.2$ for step $itermax$. In general, parameter setting results robust and intuitive to tune. For the k-means algorithm, we set $k = 100$, number of iterations smaller than $1,000,000$ and, to avoid the initial centroid placement bias, we execute 10 independent runs and select the one with the best Sum of Squared Errors (SSE) [12].

A. Iterative clustering performance: We first evaluate the benefits of the iterative clustering procedure in SeLeCT when compared to the clustering algorithm that was proposed in [8] which is based on the classical k-means[3]. Experiments here consider, for each dataset, the first batch of $10,000$ flows only. The labeling process uses the DPI labels; the majority label is extended to all flows in a cluster (again, mimicking [8]). Once flows in each cluster have been labeled, the overall accuracy is computed by comparing the assigned labels versus the original DPI labels. Fig. 4 reports results for all datasets. It highlights the benefit of the iterative clustering process for which the accuracy is about 97.5% on average, with a worst case of 94.2% for DS-3C. The simple k-means adopted in [8] results in no more than 85% accuracy, which is in line to the findings in [8]. In some scenarios, the iterative clustering we propose boosts clustering homogeneity from 80% up to 99.5% (e.g., for DS-1S).

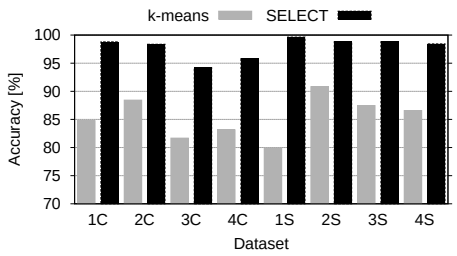

Fig. 4. Accuracy of the clusters for classic k-means and SeLeCT.

An interesting observation in Fig. 4 is that the Server datasets show better accuracy than the Client datasets. The intuition is that server responses have more peculiar lengths than client queries, e.g., HTTP responses are different from SMTP banners length.

Investigating further the reason for such poor performance of the k-means algorithm, we have observed that it tends to suffer from *capture* effects, i.e., samples of classes with small number of elements are not identified and are mixed into the clusters of the most popular classes. SeLeCT instead is able to filter out outliers from port-dominated cluster. These outliers are then successfully processed in the next phases to form clusters of less predominant classes (see [15] for more details).

Furthermore, we observed that SeLeCT is more robust than the DPI-based classifier because layer-4 features are less sensitive to small feature changes that the DPI pattern matching rules. For example, a simple character change in the protocol signature was able to fool the DPI which mislabeled some SMTP flows as unknown in the DS-2 where the SMTP banner included some non-English pattern.

B. Interesting findings enabled by SeLeCT: We investigate clusters whose DPI inherited label is "Unknown" for all datasets. In more details, SeLeCT identified the following clusters among "Unknown" flows[4] used as oracles:

- $srvPort = 1755$ - the **Microsoft Media Server (MMS)** protocol is found in DS-1;
- $srvPort = 1863$ - the **Microsoft Messenger (MSN)** protocol is found in all datasets;
- $srvPort = 1935$ - the **Macromedia RTMP** protocol is found in DS-3 and DS-4;
- $srvPort = 5223$ - the **Apple push notification server over TLS** is found in DS-3 and DS-4;
- $srvPort = 5152$ - **Backdoor.Laphex.Client** traffic is found in DS-1;
- $srvPort = 12350$ - the **Skype proprietary authentication** protocol is found in DS-3 and DS-4;

Label correctness for the first three clusters has been confirmed by manually inspecting the flow payload in each

[2]http://www.narus.com/index.php/product/narusinsight

[3]We tested other clustering algorithms like DB-SCAN or x-means. Results are similar or worse, with a trickier sensitivity to parameter settings.

[4]Some DPI signatures are present in the DPI tools, but revealed to be ineffective due to the nature of the traces, e.g., asymmetry in routing, or different patterns due to non-English customization of protocols.

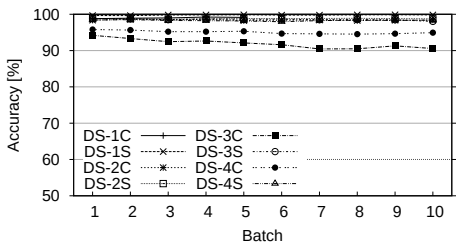

Fig. 5. Accuracy over different batches.

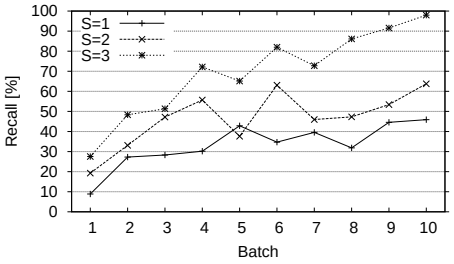

Fig. 6. eMule recall when only S labeled clusters are used at bootstrap.

cluster. For these protocols, DPI would be a viable solution if i) correct and ii) properly customized to the monitored network signatures were provided. Even more appealing is the ability of SeLeCT to automatically reveal traffic generated by novel services that appear as real unknown to the network administrator. This is the case of the *Apple Push Notification* system for iOS and iCloud enabled devices, which is based on the SSL/TLS protocol, but running on $srvPort = 5223$. All flows in this cluster are labeled by the DPI as SSL/TLS protocol. To find the correct label, a *whois* lookup for the $srvIP$ addresses reveals that the servers are all registered to Apple Inc. Active experiments confirm that this cluster is related to Apple Push Notification and iCloud services.

A second cluster of unknown flows aggregates traffic generated by the malware *Backdoor.Laphex.Client Bot/Trojan*. Manual inspection of flows payload confirms this assumption. Finally, a cluster of flows directed to $srvPort = 12350$ turns out to unveil *Skype Authentication* protocol traffic. Also in this case, the $srvIP$ reveals strong clues about the application. All flows are directed to $srvIP$ in the subnet 213.146.189.0/24, registered to Skype Inc.

Overall, we were able to find labels for about 90% of unlabeled clusters. The remaining 10% of clusters contains flows that appear to be encrypted, and for which the IP addresses point to ranges dynamically assigned to customers hosts/modems by ISPs. We suspect those could be Skype flows, but we are not able to confirm this assumption.

These examples confirm the ability of SeLeCT to automatically reveal new classes of traffic that would be hard to highlight by means of any supervised technique. Once SeLeCT is augmented with this knowledge by injecting these labels, flows are correctly classified in all subsequent batches thanks to the seeding mechanism.

V. Exploring the seeding process

We are interested now in analyzing the performance of the seeding process. We run SeLeCT on ten successive batches of flows. As previously done, the bootstrapping at batch 1 is initialized using the DPI labels. Then, for the subsequent batches, **extractSeeds()** is used to seed the labeling process from batch n to batch $n + 1$.

A. Self-seeding: Fig. 5 shows the results for all datasets. First, notice how the accuracy of SeLeCT is very high and stable over time for all server datasets. For DS-3C and DS-4C, the accuracy slightly decreases over time. For instance, in DS-3C it decreases to about 90% during the first 7 batches, then it stabilizes. Investigating further, we notice that both recall and precision of SeLeCT are permanently higher than 98% for all classes of traffic except for BitTorrent and eMule protocols. This is a desirable property, since confusion actually happens among P2P protocols only. The relative higher fraction of P2P traffic in the DS-3C (collected at 2am) results in a global decrease in the overall accuracy. For the DS-4C (referring to peak time), the fraction of P2P flows is smaller than during the night and thus it has less impact on the overall accuracy.

Finally, for DS-4S, the scenario with the highest flow arrival rate, SeLeCT was able to complete the processing of batch n before the collection of flows of batch $n + 1$ was complete, thus enabling real-time operation. While we acknowledge that more thorough testing and implementation should be done to validate the scalability of SeLeCT, this result confirms that the processing time required by SeLeCT is small enough to be considered for live deployment.

B. Bootstrapping: As we noted before, SeLeCT requires manual intervention to provide labels to clusters. When a label for a few flows is introduced, SeLeCT will carry it on for future classification. We now investigate how difficult it can be to manually bootstrap the system. We assume that a network operator is offered clusters of flows, and s/he has to use her/his domain knowledge to provide labels.

We consider the DS-4S trace, ignore all the DPI labels, and provide no labels to SeLeCT. At the end of the first batch, the operator has to analyze the clusters that have been formed to label them.

To assign a label, the information provided by the $srvPort$ for *dominatedPort* clusters provides to be very valuable. In addition to clusters dominated by well-known ports (whose label is trivial to assign), SeLeCT naturally creates some clusters whose protocol was not even known to the DPI, e.g., a cluster dominated by port 5223 that is used by Apple push notification services, or port 12350 cluster that contains flows going to Skype Inc. managed servers. As we have seen in the previous section, labeling dominatedPort clusters is very easy.

The analysis of *randomPort* clusters is expected to be more complicated since the $srvPort$ information is, by construction, providing limited information.

Nonetheless, $srvPort$ analysis still provides vital clues about the protocol when analyzing the port number frequency distribution by considering all flows in a cluster together. For instance, consider a P2P protocol in which the user can manually change the port used by the application, e.g., eMule. It is very likely that the port the user would choose is “similar”

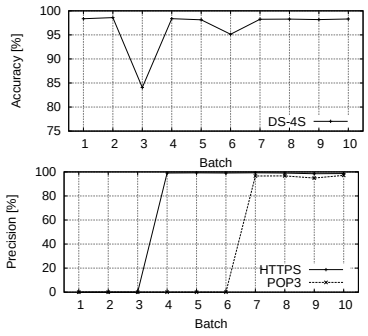

Fig. 7. New protocols suddenly appear: HTTPS traffic is added at batch 3, and POP3 traffic is added at batch 6 in DS-4S.

to the default number offered by the application, therefore biasing the port frequency distribution. Consider a cluster in which the topmost ports are 4664, 4661, 8499, 7662, 6662, 5662, 4663, 64722, ... The intuition suggests to label flows in that cluster as eMule whose default port is 4662 (which turns out to be the correct label). On the contrary, clusters in which port numbers are uniformly distributed clearly suggest that the application itself is enforcing a random port selection, as done, e.g., by most popular BitTorrent applications. Some domain knowledge coupled with ingenuity allows to easily assign a proper label also in this case.

In general, and unlike traditional per-flow analysis, the inspection of clustered flows simplifies the identification of labels since all flows in a cluster share a common threat and thus can be analyzed in parallel.

C. Seeding evolution: To show the ability of SeLeCT to increase its knowledge over time, we perform the following experiment. Consider DS-4S and focus on the eMule flows not having the default 4662 $srvPort$ (which are clustered as $dominatedPorts$ clusters). At the end of batch 1 processing, only the largest S $randomPort$ clusters are manually labeled as eMule (e.g., by checking the port number distribution as above). Labeled flows are then used to bootstrap the seeding process. Fig. 6 reports the recall evolution over the different batches for different values of S. For $S = 3$, SeLeCT already achieves 98% of recall at batch 10. Worst case precision is 98.6%. Notice that the seeding process is successfully bootstrapped even if only $S = 1$ cluster is used as initial seed, even if with slower convergence time.

We now perform another experiment in which we simulate the sudden appearance of a new class of traffic. We consider the DS-4S trace, from which we removed all POP3 and HTTPS flows. Then, during the third and sixth batch, HTTPS and POP3 traffic is injected to simulate the sudden birth of new protocols. We run SeLeCT over all 10 batches. Results are reported in Fig. 7. The top plot reports the overall accuracy, while the bottom plot reports precision, respectively. Notice how SeLeCT rapidly detects the presence of new traffic classes. In particular, at batch 3, accuracy severely drops since HTTPS flows are mislabeled as "Unknown". At batch 4, HTTPS bootstrapping is performed, and accuracy returns to 97.5%, and HTTPS precision and recall approach 100%.

The same transient is observed when POP3 flows are injected. Being their number small, the impairment on accuracy is less evident. From batch 7 on, the bootstrapping of the POP3 protocol is completed so that accuracy, recall (and precision) get back to excellent values.

These examples show how easy is to augment network administrator visibility by providing homogeneous clusters of flows whose analysis is much easier, due to the aggregated information provided by the flows in the cluster.

VI. Conclusions

In this paper we presented SeLeCT, a semi-automated Internet flow traffic classifier which leverages unsupervised clustering algorithms to automatically groups flows into clusters. SeLeCT significantly improves the performance and coverage of previous proposal by alternating clustering and filtering phases. Cluster labeling is easily bootstrapped using several approaches, including simplified human-in-the-middle. Once labels for some flows are provided, SeLeCT inherits them to automatically label new clusters. Furthermore, it adapts the model to traffic changes, and it is able to increase its knowledge.

Extensive experiments showed that SeLeCT achieves excellent performance: Accuracy is close to 98% in most datasets, with worst case still higher than 90%. SeLeCT proved able to automatically expose classes of traffic that even an advanced DPI-based classifier was ignoring.

References

[1] T. Nguyen, G. Armitage. A Survey of Techniques for Internet Traffic Classification using Machine Learning. *IEEE Communications Surveys and Tutorials*, vol. 10, no. 4, 2008.

[2] H.Kim, KC.Claffy, M.Fomenkov, D.Barman, M.Faloutsos, K.Lee. Internet traffic classification demystified: myths, caveats, and the best practices. *ACM CoNEXT*, Madrid, SP, 2009.

[3] T. Karagiannis, D. Papagiannaki, M. Faloutsos. Blinc: Multilevel traffic classification in the dark. *ACM SIGCOMM*, Philadelphia, PA, 2005.

[4] J. Erman, M. Arlitt, A. Mahanti. Traffic classification using clustering algorithms. *ACM SIGCOMM*, Pisa, IT, 2006.

[5] A. Demiriz, K. Bennett, M. Embrechts. Semi-supervised clustering using genetic algorithms. *ANNIE 99*, St. Louis, MO, 1999.

[6] R. Dara, S.C. Kremer, D.A. Stacey. Clustering unlabeled data with SOMs improves classification of labeled real-world data. *IEEE IJCNN*, Honolulu, HA, v.3, pp.2237/2242, 2002.

[7] A. McGregor, M. Hall, P. Lorier, J. Brunskill. Flow clustering using machine learning techniques. *PAM 2004*, Antibes, FR, 2004.

[8] J.Erman, A.Mahanti, M.Arlitt, I.Cohen, C.Williamson. Offline/realtime traffic classification using semi-supervised learning. *Perform. Eval.*, v.64, n.9-12, pp.1194-1213, 2007.

[9] A.Finamore, M.Mellia, M.Meo. Mining unclassified traffic using automatic clustering techniques. *TMA 2011*, Vienna, Austria, 2011.

[10] L. Bernaille, R. Teixeira, K. Salamatian. Early application identification. *ACM CoNEXT*, Lisboa, PT, 2006.

[11] Y. Wang, Y. Xiang, S. Yu. An automatic application signature construction system for unknown traffic. *Concurrency and Computation: Practice and Experience 2010*, vol.22, pp.1927-1944, 2010.

[12] P.N. Tan, M. Steinbach, V. Kumar, others. Introduction to data mining. *Pearson Addison Wesley Boston*, 2006.

[13] I.Trestian, S.Ranjan, A.Kuzmanovic, A.Nucci, Googling the Internet: Profiling Internet Endpoints via the World Wide Web. *IEEE/ACM Transactions on Networking*, v.18, n.2, pp.666-679, 2010.

[14] A.Finamore, M.Mellia, M.Meo, M.Munafo, D.Rossi, "Experiences of Internet traffic monitoring with tstat," *Network, IEEE*, vol.25, no.3, 2011.

[15] L.Grimaudo, M.Mellia, E.Baralis, R.Keralapura, "TR-291112 - SELECT: Self-Learning Classifier for Internet Traffic", submitted to IEEE Transactions on Networking, http://www.tlc.polito.it/mellia/TR-291112.pdf.