SeLeCT: Self-Learning Classifier for Internet Traffic

SeLeCT: Self-Learning Classifier for Internet Traffic

Luigi Grimaudo, Marco Mellia, Elena Baralis and Ram Keralapura

***Abstract*—Network visibility is a critical part of traffic engineering, network management, and security. The most popular current solutions - Deep Packet Inspection (DPI) and statistical classification, deeply rely on the availability of a training set. Besides the cumbersome need to regularly update the signatures, their visibility is limited to classes the classifier has been trained for. 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.**

To address the above issues, we propose SeLeCT, a Self-Learning Classifier for Internet Traffic. It uses unsupervised algorithms along with an adaptive seeding approach to automatically let classes of traffic emerge, being identified and labeled. Unlike traditional classifiers, it requires neither a-priori knowledge of signatures nor a training set to extract the signatures. Instead, SeLeCT automatically groups flows into pure (or homogeneous) clusters using simple statistical features. SeLeCT simplifies label assignment (which is still based on some manual intervention) so that proper class labels can be easily discovered. Furthermore, SeLeCT uses an iterative seeding approach to boost its ability to cope with new protocols and applications.

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 excellent precision and recall, with overall accuracy close to 98%. Unlike state-of-art classifiers, the biggest advantage of SeLeCT is its ability to discover new protocols and applications in an almost automated fashion.

***Index Terms*—Traffic classification, clustering, self-seeding, unsupervised machine learning**

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). Until a decade ago, *port-based* approaches were very popular. The effectiveness of pure port-based approach has diminished even if it has been shown that port numbers carry valuable information about the application and/or protocol [2]. Over the last few years *deep packet inspection* (DPI) has become popular [1], and *behavioral* techniques have been investigated since the seminal work of [3].

Both DPI and behavioral classifiers share some limitations. First, to achieve a high classification accuracy, either a cumbersome protocol reverse engineering to identify the signatures in DPI, or a tedious process to generate an accurate training set for behavioral classifiers is required. In other words, both approaches require *training*. Second, and most critical, the classifiers *can identify only the specific applications they have been trained for*. All other traffic is aggregated either in a generic class labeled as "unknown", or, even worse, it is mislabeled as one of the known applications. In other words, these classifiers cannot identify the introduction of a new application, or changes in the applications' protocol or behavior, unless a 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. Our goal is to provide a deeper network visibility for operators. In other words, we intend to offer the ability to semi-automatically identify prominent classes of traffic, targeting network management and traffic engineering operations[1]. SeLeCT proves to be able to expose classes of traffic which are very specific and possibly are not already known to the operator. For example, SeLeCT has been able to separate Google Mail traffic from other mail services. It thus automatically allows to discover new classes of traffic, allowing arbitrary definition of labels.

In SeLeCT, we leverage unsupervised data mining algorithms to automatically split traffic into homogeneous subsets or clusters. We consider *flows* as the target of the classification. Each flow is characterized by using simple layer-4 metrics, like segment size and inter-arrival time. These features are known to carry valuable information about the protocol and/or application that generated the flow [2]. However, they perform not as good in the context of unsupervised (i.e. clustering) algorithms. Hence we have to adopt some ingenuity in order to improve cluster homogeneity. To overcome the limitation of off-the-shelf algorithms, we design an *iterative clustering* procedure in which a filtering phase follows each clustering phase to eliminate possible outliers. Filtering is based on the still valuable information provided by port numbers. Note that port number information is not embedded in a metric space,

Manuscript received June 10, 2013; revised December 20, 2013; handled by associate editor Dr. Philippe Owezarski.

The research leading to these results has been partly funded by the European Union under the FP7 Grant Agreement n. 318627 (Integrated Project "mPlane") and by Narus Inc.

Luigi Grimaudo, Marco Mellia, and Elena Baralis are with Politecnico di Torino, Italy - email: firstname.lastname@polito.it

Ram Keralapura is with Narus Inc, Sunnyvale, California. E-mail: rkeralapura@Narus.com.

[1]SeLeCT is *not* intended for security purposes where every single bit, packet, and/or flow must be carefully examined.

e.g., the distance between port 79 and 80 is not different from the one between port 80 and 8080. As such, it is hard to integrate port number as a simple feature into classical clustering algorithms.

Using traffic traces collected in different years from various ISPs located in 3 different continents, we show that the iterative clustering process leads to clusters with excellent properties. First, SeLeCT generated only a few cluster in each of these traces (typically less than 150). Second, clusters are very pure, i.e., the overall homogeneity of the clusters is close to 100%. This allows to easily inspect and label each cluster, thus assigning a proper label to all flows belonging to the same cluster.

As soon as some labels are assigned to flows, SeLeCT will automatically inherit them for classification of flows that arrive in the future. We refer to this as *adaptive* or *progressive* seeding 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 mainly required for the initial label assignment. This mechanism allows to naturally grow the intelligence of the system such that it is able to automatically adapt to the evolution of protocols and applications, as well as to discover new applications.

The idea of leveraging semi-supervised learning has been initially proposed in [4], where the authors leverage the standard k-means to construct clusters. Part of the flows to be clustered are assumed to be already labeled, and a simple voting scheme is used to extend the dominant label to the whole cluster.

SeLeCT follows similar principles, extending the idea with i) iterative port filtering and ii) multi-batch seeding which, as we will see in Sec. VI, allow to significantly boost overall performance achieving 98% accuracy in practical cases. The iterative clustering algorithm and self-seeding approach provide several advantages: the number of clusters is reduced to less than 150, while at the same time homogeneity is significantly increased. This simplifies the labeling process so that manual inspection becomes almost trivial. Furthermore, the SeLeCT self-seeding process is more robust and results obtained from actual traffic traces show how SeLeCT helps in automatically identifying *fine grained classes of traffic* (e.g, IMAP vs POP3, XMPP vs Messenger), and even unveiling the presence of unknown/undesired classes (e.g., Apple push notification, Bot/Trojan, or Skype authentication traffic). In identifying standard protocols SeLeCT proves to be even more robust than professional DPI based tools which were fooled by non-English customizations of protocol error messages.

A preliminary version of this paper appeared in [5]. This version improves description of the algorithms and a complete performance evaluation of SeLeCT including a thorough sensitivity analysis, and more detailed set of results.

A. Key features of SeLeCT

Some of the key features of SeLeCT are:

• *Adaptive classification model.* A semi-supervised learning approach allows SeLeCT to learn information from unlabeled data with simplified manual intervention. Once some labels are provided, SeLeCT automatically adapts the model to changes in the traffic.

• *Simple iterative approach.* SeLeCT is based on k-means, a simple yet effective clustering algorithm. It uses k-means as a building block in an iterative clustering refinement process, which allows leveraging specific Internet traffic features such as the server port that cannot be integrated into classical clustering algorithms in a straightforward fashion. This approach yields strongly cohesive clusters and provides an almost complete coverage of the considered flows.

• *Leverages layer-4 features.* SeLeCT relies on the availability of flow level features that can be easily acquired at the beginning of the flow, and it does not assume to see both directions of traffic.

• *Limited complexity.* SeLeCT can run in real time by constantly monitoring the incoming traffic, creating batches of flows, and processing these batches before the next batch accumulates.

After describing related works in Sec. II, we introduce the classification problem in Sec. III and provide a description of the datasets used for the experiments in Sec. IV. SeLeCT algorithms are detailed in Sec. V, while performance evaluation is provided in Sec. VI. Interesting findings are highlighted in Sec. VII, while Sec. VIII discusses the self-seeding process. Sec IX discusses parameter setting and provides a thorough sensitivity analysis. Finally, Sec. X summarizes the key findings.

II. Related Work

A. Clustering Algorithms

Data mining techniques may be grouped in two families: *supervised* and *unsupervised* techniques [6]. Supervised algorithms assume the availability of a training dataset in which each object is labeled, i.e., it is a-priori associated to a particular class. This information is used to create a suitable model describing groups of objects with the same label. Then, unlabeled objects can be classified, i.e., associated to a previously defined class, according to their features. For unsupervised algorithms, instead, grouping is performed without any a-priori knowledge of labels. Groups of objects are clustered based on a notion of distance evaluated among samples, so that objects with similar features are part of the same cluster.

Supervised algorithms achieve high classification accuracy, provided that the training set is representative of the objects. However, labeled data may be difficult, or time-consuming to obtain. Semi-supervised classification addresses this issue by exploiting the information available in unlabeled data to improve classifier performance. Many semi-supervised learning methods have been proposed [7], unfortunately, no single method fits all problems.

The semi-supervised learning approaches closest to our proposal are [8] and [9]. Both labeled and unlabeled data are clustered by means of (variations of) known clustering algorithms (k-means in [8] and SOM in [9]). Next, labeled data in each cluster is exploited to assign labels to unlabeled data.

Finally a new classifier is trained on the entire labeled dataset. While we exploit a different, iterative clustering approach to group data, our labeling process is similar to [8]. Due to its iterative refinement process, the approach adopted in SeLeCT is also particularly suited to model traffic flow changes, because it allows a seamless adaptation of the obtained traffic classes to traffic pattern evolution.

B. Applications to traffic classification

The application of unsupervised techniques is not new in the traffic classification field. [10] is one of the preliminary works and shows that clustering techniques are useful to obtain insights about the traffic. In [11] supervised and unsupervised techniques are compared, demonstrating that unsupervised algorithms can achieve performance similar to the supervised algorithms. Other works compare the accuracy of different and standard unsupervised algorithms [12], [13], [14]. In general, the techniques presented in these works achieve a moderate accuracy and they typically identify several hundreds of clusters, therefore questioning the applicability of this methodology in practice.

Recently, [4], [15], [16], [17], [18] have introduced the *semi-supervised* methodology in the context of traffic classification. [4] is among the first works that proposes also a simple labeling algorithm. It uses the off-the-shelf k-means algorithm and present a performance evaluation considering a trace collected from a Campus and a small residential network. Limited ground truth is available and only coarse classes are considered (e.g., P2P, HTTP, EMAIL, CHAT, etc.). Results show that to achieve good accuracy, a still large number of clusters must be used ($k \geq 400$) and the labeled dataset must be large (more than 15% of flows must be already labeled). We explicitly compare the performance of SeLeCT against the solution proposed in [4] in Sec. VI.

In [15] the authors propose a simple clustering algorithm based on information entropy to group flows. Clusters are then labeled using some ad-hoc engineered algorithm that can coarsely identify classes like P2P or Client/Server traffic. Limited performance evaluation is provided considering traffic generated by 20 hosts only. Neither learning nor seeding is proposed. In [16], the authors proposed advanced unsupervised and semi-supervised machine learning algorithms to cluster flows. 22 (bi-directional) flow level features are used, which include packet size and inter-arrival time. Performance evaluation considers two small datasets of 4,000 flows each. Accuracy reaches 85%. [17] proposes a semi-supervised method which extends [4]. As features, the destination IP address, server port and transport protocol are considered. k-means is used as basic building block. Accuracy, evaluated considering two traffic traces, tops 90%. In [18], the authors propose an unsupervised traffic classification that uses both flow features and packet payload. Using a bag-of-words approach and latent semantic analysis, some clusters are identified. Performance is evaluated using a single trace and reaches 90% of accuracy.

In all cases, SeLeCT achieves better results in terms of classification performance, provides finer grained visibility on traffic, and offers a simple self-seeding mechanism that naturally allows the system to increase its knowledge..

III. Problem statement

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. For instance, in a traditional TCP connection, packets sent by the client belong to a directed flow, and packets sent by the server belong to a different flow. Considering directed flows 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. A flow F is identified when the first packet is observed; the flow ends when no packets have been seen for a given time ΔT. TCP signaling segments may be used to detect appropriate flow start and end. As suggested in [19], we consider a conservative value of $\Delta T = 5min$.

For each flow F, a set of features $A(F) = \{a_1^{(F)}, a_2^{(F)}, \ldots, a_n^{(F)}\}$ is collected. These features are used by SeLeCT to characterize flows and take the classification decision. The goal of SeLeCT is to assign a proper application to each flow F based on the sole knowledge of the flow feature set $A(F)$.

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. Note that only flows that have more than n segments can be classified. The impact of the choice of n is discussed in IV.

Formally, let $L(i_F)$ be the length of the i-th segment of flow F, and let $t(i_F)$ be its arrival time. The i-th inter-arrival time $\Delta t(i_F)$ is $\Delta t(i_F) = t(i_F) - t(i_F - 1),\ i_F > 1$. Then

$$A(F) = \{srvPrt,\ L(i_F),\ \Delta t(i_F)\ \forall i_F \leq n \wedge L(i_F) > 0\}$$

The choice of which features to consider is a matter of optimization and several works in the literature have proposed and investigated possible alternatives. Our choice stems from the following intuitions: (i) keep the feature set limited, (ii) include generic layer-4 features that can be easily computed, and (iii) use features that can be collected during the beginning of a flow so that we can classify flows in real-time (i.e., minimize the time required for identification). It is out of the scope of this paper to compare and choose which are the most suitable features to use. We will consider this as a part of our future work. However, given the high accuracy of SeLeCT, we believe that it may be difficult to improve it by considering a wider/different set of features.

IV. Datasets to evaluate SeLeCT

In this section, we briefly describe the datasets that we collected and used to evaluate SeLeCT. We provide more details in Section VI. Table I summarizes the main characteristics of the datasets. We collected four different traces

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

TABLE I
DATASETS USED IN THE PAPER FOR PERFORMANCE EVALUATION. THE TABLE INCLUDES FLOWS FOR WHICH FEATURES CAN BE COMPUTED.

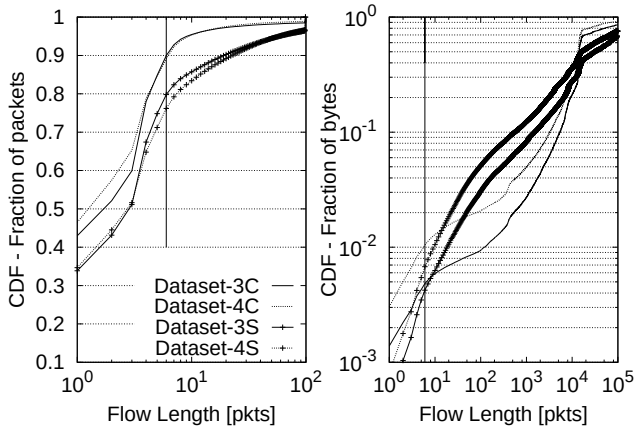


Fig. 1. CDF of the flow length in packets (on the left), and bytes (on the right). The vertical line is in correspondence of 6 data packets.

from access and backbone networks of large ISPs[2]. Each dataset is a 1-hour long complete packet trace including the packet payloads. We selected these traces to create a very heterogeneous benchmarking set. They include backbone and access scenarios, day and night time periods, different years, and users from three different continents.

In this work, we focus on TCP traffic only as most applications today rely on TCP. The extension of SeLeCT to UDP traffic is straightforward and is not further investigated in this paper.

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. Overall, the oldest trace - Dataset-1 - was collected in 2005 from a major ISP in South America; it contains more than half million TCP flows involving more than $100,000$ hosts. Dataset-2 was collected from the peering link of an ISP in Asia in September 2010. Finally, Dataset-3 and Dataset-4 were collected at different times of the day from the same vantage point in Europe during August 2011. Dataset-3 was collected at 2am in the night, while Dataset-4 was collected at 5pm. The latter contains about 2.3 million flows directed to more than $190,000$ hosts. We will primarily use the last two datasets for deeper investigation in the rest of the paper.

Only flows that have at least n data packets can be considered by SeLeCT. So the first question to answer is how much traffic can be classified by SeLeCT for different values of n. Fig. 1 reports the Cumulative Distribution Function (CDF) of the number of packets (on the left) and bytes (on the right) carried by flows of different length. For the sake of simplicity, let us focus on Dataset-3 and Dataset-4, which are the two most recent datasets. The CDF of the fraction of packets (on the left plot) shows that 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 (highlighted by the vertical bar). However, by looking at the CDF of bytes (reported on right plot), we observe that the mice account for no more than 1% of the volume of traffic (notice the log scale on y-axis). 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 $n = 6$. Thus, as any statistical classifier, SeLeCT targets long-lived flows.

[2]Due to NdA with ISPs we are not allowed to share the original traffic traces.

```
1: Main()
2: Output: set C of labeled clusters
3: S = ∅
4: while (newbatch B) do
5:    ProcessBatch(B, U, S, C, NS)
6:    S = NS
7: end while
8:
9: ProcessBatch(B, U, S, C, NS):
10: Input: Set B of new flows, set S of seeds
11: Output: set C of labeled clusters, set NS of new seeds
12: B' = B ∪ S ∪ U /* Merge new flow, seeding set,
13:                   and past outliers */
14: C' = doIterativeClustering(B');
15: C = doLabeling(C');
16: NS = extractSeeds(C);
```

Algorithm 1: SeLeCT Main loop.

V. 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*. A new batch $\mathcal{B}$ is formed as soon as a given number of valid flows is observed. The probe monitors packets and rebuilds flows. For a given flow, as soon as 6 data packets are observed the flow identifier and features are dispatched to a buffer where the batch is being formed. When the batch reaches the target number of flows, it is dispatched to the classification algorithm, and a new batch starts.

SeLeCT analyzes each batch of newly collected flows via the **ProcessBatch()** function shown in the pseudo-code reported in Alg. 1. This function takes in input

- $\mathcal{B}$, the batch of new flows;
- $\mathcal{U}$, the set of previous outliers that were not assigned to any class when processing the previous batch;
- $\mathcal{S}$, the set of *seeding flows*, i.e., flows already analysed in past batches for which SeLeCT was able to provide a

label;

As output, it produces

- $\mathcal{C}$, the set of clusters;
- $\mathcal{NS}$, the set of new seeds that are extracted from each cluster;
- $\mathcal{U}$, which contains the set of new outliers;

Its main steps (see Alg. 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()**). Note that flows that are not assigned to any cluster are returned in the $\mathcal{U}$ set. Those flows are then aggregated in the next batch, so that they can eventually be aggregated to some cluster[3]. In the following we detail each step of the batch processing.

A. Iterative clustering

Clustering algorithms group objects with similar characteristics [6]. Objects are described by means of features which map each object to a specific position in a hyperspace. The similarity between two objects is based on their *distance*. The closer the two objects are, the more likely they are similar and thus should to be grouped in the same cluster. Typically, the Euclidean distance is used.

Iterative clustering is the core of SeLeCT. It exploits the k-means clustering algorithm [6] to group flows into subsets or *clusters* which are possibly generated by the same applications. We selected the k-means algorithm since it is well understood and it has been previously used in previous works. We tested also other clustering algorithms like DBSCAN [6]. Results (not reported here due to lack of space) are similar or worse, with a trickier sensitivity to parameter settings.

In this context, it is natural to consider two flows with similar packet lengths and inter-arrival times 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 [6], 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*.

In the past, several researchers have applied clustering algorithms to traffic analysis [4], [12]. However, to the best of our knowledge, none of the previous works exploited the specific characteristic of the $srvPort$ feature in a clustering process. This is mainly related to the fact that port numbers are not embedded in a metric space. Thus ingenuity is required to smartly include them. In our work we engineer an iterative procedure to identify clusters of flows in which the $srvPort$ information is used to *filter* elements in each cluster. As reported in Alg. 3, we devise an iterative process, in which clustering phases and filtering phases alternate. We use a set-based notation. Names of the sets are defined in the pseudo code.

[3]It would be possible to limit the number of batches some flows may be still in the $\mathcal{U}$ set and output them in a "unclassifiable" set to avoid delaying classification process.

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

Algorithm 2: Filtering of clusters.

1) The filtering procedure: The filtering procedure is reported in Alg. 2. Filtering is performed on the cluster $\mathcal{I}$ provided as input. First, **doFiltering()** discards clusters which have less than $minPoints$ flows to avoid dealing with excessively small clusters. Discarded flows are returned in set $\mathcal{U}$, the set of unclustered flows that will undergo a subsequent clustering phase (lines 5-7).

$DominatingPhase$ is a flag that is used to select the type of filtering: when it is TRUE, the filtering processes only *dominatedPort* clusters. 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: The iterative clustering procedure is reported in Alg. 3 which receives as input

1: **doIterativeClustering**($\mathcal{B}$)
2: *Input*: Set $\mathcal{B}$ of flows to be clustered
3: *Output*: set of clusters $\mathcal{C}$, set of outliers $\mathcal{U}$
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: $\mathcal{U} = \emptyset$
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: /* Last step: process random port clusters */
14: **for** dp in $\mathcal{DP}$ **do**
15: delFlows($\mathcal{U}$,dp) /* Discard flows still to $\mathcal{DP}$ */
16: **end for**
17: $\mathcal{C}'$ = k-means($\mathcal{U}$)
18: **for** $\mathcal{I}$ in $\mathcal{C}'$ **do**
19: /* look for *randomPort* clusters now */
20: doFiltering($\mathcal{I}$,$\mathcal{C}$,$\mathcal{U}$,$\mathcal{DP}$,0,false)
21: **end for**
22: return $\mathcal{C}$, $\mathcal{U}$

Algorithm 3: Iterative Clustering

the current batch $\mathcal{B}$ of flows. It iteratively generates dominated port clusters alternating clustering and filtering phases. At last, it generates random port clusters. More specifically, the set of flows $\mathcal{B}$ 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 that returns the set $\mathcal{C}'$ of k clusters. Each cluster in $\mathcal{C}'$ undergoes a filtering phase (lines 8-11), which is looking for *dominatedPort* clusters at this stage. The **doFiltering()** procedure returns in $\mathcal{U}$ flows that did 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. The set $\mathcal{DP}$ of dominant ports 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. As such, we first remove from the set $\mathcal{U}$ of flows to be clustered all those flows directed to any dominating port that has been found in the previous iterations (lines 14-16). Then, the final clustering is completed (line 17-21).

B. Labeling

Once flows have been clustered, the **doLabeling**($\mathcal{C}'$) procedure (see Alg. 1 - line 15) 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 (extracted from $\mathcal{S}$) that already have a label, a simple majority voting scheme is adopted: the seeding flow 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 as shown in Sec. VIII.

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. VIII. 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, (as in [4]).

In all cases, the complexity of the labeling process is reduced to the analysis of few clusters, instead of hundred of thousands of flows. This mechanism can be also automated as suggested by [20], 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 (see Alg. 1 - line 16). It implements a *stratified sampling technique*, i.e., from each cluster, the number of extracted seeds is proportional to the cluster size. Stratified sampling ensures that at least one observation is picked from each of the cluster, even if probability of it being selected is far less than 1. Thus, it guarantees that in the seeding set there are representatives of each cluster and avoids the bias due to classes having much more flows than others. Let $numSeeds$ be the target number of seeding flows, i.e., $numSeeds = ||\mathcal{NS}||$. For each labeled cluster $\mathcal{I}$, a number NS_I of labeled flows proportional to the cluster size is extracted at random. That is $NS_I = 1 + \left(\frac{||\mathcal{I}||}{numSeeds}\right)$ flows are randomly selected from each cluster $\mathcal{I}$. 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. Sec. VIII provides some evidence to support this statement.

VI. Experimental results

A. Experimental dataset

We performed several experiments to assess the performance of SeLeCT using the datasets described in Sec. IV. All traces have been processed to generate directed flow level logs. Recall that we only consider TCP flows in this work. We use two separate advanced DPI classifiers to label flows and use these labels as our ground truth. The first one

is provided by the NarusInsight[4] professional tool, and the second one is implemented in Tstat [21], the Open Source traffic monitoring developed at Politecnico di Torino. 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). To be conservative, we label as "unknown" those flows that do not match any of the DPI rules, or for which DPIs' labels are different. 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.

B. Performance metrics

We consider two metrics to characterize the output of the iterative clustering algorithm: *number of clusters* and *clustered flows percentage* (i.e., the ratio of flows $||\mathcal{C}'||$ clustered by **doIterativeClustering($\mathcal{B}'$)** to the total number of flows $||\mathcal{B}'||$ provided as input expressed in percentage).

In order to evaluate classification performance, we use the *confusion matrix*. The confusion matrix is a matrix in which each row represents the instances in a predicted class (i.e., the decision of SeLeCT), while each column represents the instances in an actual class (i.e., the ground truth). The name stems from the fact that it highlights cases in which the system is confusing two classes (i.e., it is mislabeling one as another). To evaluate the classification performance of SeLeCT, we use three metrics: *overall accuracy, recall,* and *precision*.

• *Accuracy* is the ratio of the sum of elements in the main diagonal (i.e., the total true positives) of the confusion matrix to the sum of all elements (i.e., the total samples). Accuracy does not distinguish among classes and is biased towards dominant classes in the dataset. For instance, consider a scenario where 90% of flows are HTTP flows. A classifier that always returns the "HTTP" label will have accuracy of 90%, despite completely missing all the other classes. Although accuracy is an important metric, it does not capture all the characteristics of the classifier.

• *Recall* for the i-th class, is the ratio of the element (i, i) (i.e., the true positives) in the confusion matrix to the sum of all elements in the i-th column (i.e., the total samples belonging to the i-th class). It measures the ability of a classifier to select instances of class i from a data set. In the same example as before, always returning "HTTP" would have a recall of 0% for all classes except for "HTTP".

• *Precision*, for the i-th class, is the ratio of the element (i, i) in the confusion matrix to the sum of all the elements in the i-th row (i.e., the true positives plus the false positives). It measures the ability of the classifier in assigning only correct samples to class i. In the example above, always returning "HTTP" would have a precision of 90% for the HTTP class and of 0% for the other classes.

In the rest of this section, we consider 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 $itermax$. Extensive parameter sensitivity is carried over in Sec. IX. 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 result with the best Sum of Squared Errors (SSE) [6].

[4]http://www.narus.com/

C. Iterative clustering performance

We first evaluate the benefits of the iterative clustering procedure in SeLeCT. We compare the accuracy against i) simple port-based classifier and ii) classic k-means as proposed in [4]. The simple port-based classifiers uses the $srvPort$ to label flows. It considers well-known ports for the most common protocols, and port 4662 for eMule. Experiments here consider, for each dataset, the first batch of 10,000 flows only. For both algorithms, labeling is performed by the **doLabeling()** procedure. The labeling process adopts a simple majority voting scheme: given a cluster, the most frequent label among seeding flows in the cluster is extracted, and used to label all flows (mimicking [4]). The assigned label is then compared to the original label that the DPI assigned to each flow. Fig. 2 reports results for all datasets. Flow-wise and byte-wise accuracy are reported in top and bottom plot, respectively. The former is computed as the percentage of the correctly classified flows, while the latter is computed as the percentage of the bytes carried by correctly classified flows. Results highlight 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 Dataset-3C considering flow-wise accuracy.

The simple k-means adopted in [4] results in no more than 85% flow-wise accuracy, which is in line to the findings in [4], [12]. The port-based classifier performs poorly in some scenarios where protocols not using a well-known port. This is the case for Dataset-1C where the presence of P2P traffic is predominant.

SeLeCT is the only classifier that offers excellent results for all datasets, and considering both flow-wise and byte-wise accuracy. Given the marginal differences of the two metrics, in the following we consider only flow-wise performance indexes.

An interesting observation in Fig. 2 is that the Server datasets show better accuracy than the Client datasets. The reason is that layer-4 features carry more valuable information to differentiate between classes when considering packets sent by servers rather than by clients, e.g., the typical lengths of packets sent by HTTP and SMTP servers are different, while the client queries could be more similar. The intuition is that server responses have more peculiar lengths than client queries.

Table II shows the confusion matrix for Dataset-2S, which represents the *best case* for the k-means based classifier. The bold font highlights true positives. First, notice that the HTTP, SMTP, and Unknown classes are clearly predominant, possibly causing a "capture effect" so that other classes vanish. In fact, most of the flows of other classes are misclassified as one of these three predominant classes, impairing recall and

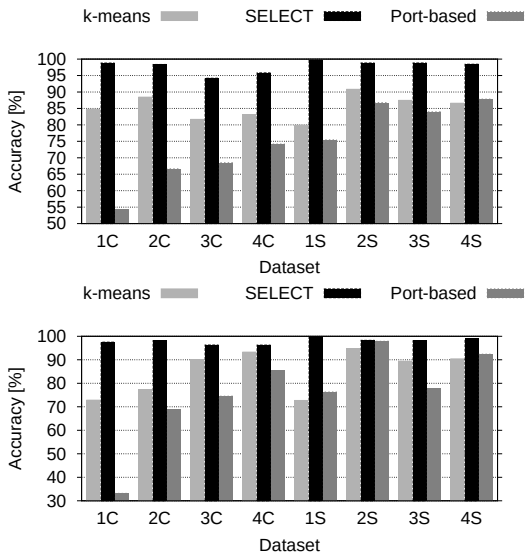


Fig. 2. Accuracy of the clusters for simple port-based classifier, classic k-means and SeLeCT. Accuracy computed per flows on the top, per byte on the bottom. Results reported for all datasets.

	BT	HTTP	HTTPS	MSN	POP3	POP3S	SMB	SMTP	SSH	Telnet	UNK	XMPP
BT	**34**	9	2	0	0	0	0	1	0	0	11	0
HTTP	18	**5829**	175	10	1	2	0	27	0	0	118	1
HTTPS	3	18	**345**	5	0	0	0	18	0	0	65	0
MSN	0	0	0	**0**	0	0	0	0	0	0	0	0
POP3	3	6	1	0	**16**	2	0	3	0	0	14	0
POP3S	0	0	0	0	0	**0**	0	0	0	0	0	0
SMB	0	0	0	0	0	0	**0**	0	0	0	0	0
SMTP	21	18	85	14	45	53	18	**2247**	0	43	276	5
SSH	0	0	0	0	0	0	0	0	**0**	0	0	0
Telnet	0	0	0	0	0	0	0	0	0	**0**	0	0
UNK	21	35	35	6	0	1	0	29	9	0	**214**	0
XMPP	0	0	0	0	0	0	0	0	0	0	0	**0**

TABLE II
CONFUSION MATRIX OF A CLASSIFIER BASED ON THE SIMPLE K-MEANS AS IN [4] CLUSTERING FOR DATASET-2S. COLUMNS GIVE THE GROUND TRUTH.

precision, even if the accuracy is still high (90% in this case - see Fig. 2). For example, POP3S and Telnet have 0% for both recall and precision. For the HTTPS flows - which are a non negligible fraction of samples - precision is 74% and recall is as low as 54%, i.e., about half of the HTTPS flows are misclassified. Finally, the predominant class performance is impaired as well. For example, SMTP precision drops to 78% because of the high number of false positives. In summary, the standard k-means clustering exhibits poor performance for not dominant classes.

SeLeCT significantly boosts performance as depicted in Table III[5]. The overall accuracy tops to 98.82% and the confusion matrix exhibits almost perfect results. Interestingly, only flows in the Unknown class have been (possibly) misclassified. For example, 102 flows that the DPI labeled as Unknown are instead labeled as SMTP by SeLeCT. We manually cross-checked these flows, and found that 97 out of 102 flows are indeed SMTP flows which the DPI was not able to correctly

[5]Totals are different than in Table II since SeLeCT adopts a conservative approach by deferring the clustering of "noise" flows to next batches.

	BT	HTTP	HTTPS	MSN	POP3	POP3S	SMB	SMTP	SSH	Telnet	UNK	XMPP
BT	**3**	0	0	0	0	0	0	0	0	0	3	0
HTTP	0	**5769**	0	0	0	0	0	0	0	0	30	0
HTTPS	0	0	**530**	0	0	0	0	0	0	0	0	0
MSN	0	0	0	**7**	0	0	0	0	0	0	1	0
POP3	0	0	0	0	**42**	0	0	0	0	0	0	0
POP3S	0	0	0	0	0	**46**	0	0	0	0	0	0
SMB	0	0	0	0	0	0	**8**	0	0	0	0	0
SMTP	0	0	0	0	0	0	0	**2217**	0	0	102	0
SSH	0	0	0	0	0	0	0	0	**9**	0	0	0
Telnet	0	0	0	0	0	0	0	0	0	**43**	0	0
UNK	4	0	0	2	0	0	0	0	0	0	**83**	0
XMPP	0	0	0	0	0	0	0	0	0	0	0	**5**

TABLE III
CONFUSION MATRIX OF THE SELECT CLASSIFIER FOR DATASET-2S. COLUMNS GIVE THE GROUND TRUTH.

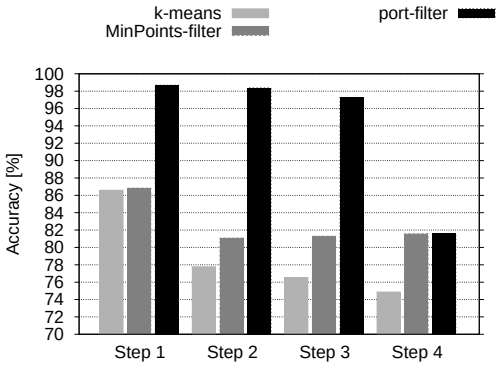


Fig. 3. Accuracy before and after the different filtering steps for Dataset-4S.

classify because the SMTP banner sent by the server was not the usual one, and its pattern was not included in the DPI engine signature set. Double checking unknown flows that SeLeCT classified as HTTP, we also verified that the DPI was fooled by some HTTP messages which included non-English text (recall this dataset was collected from an ISP in the far east). This shows that SeLeCT is able to automatically adapt classes to small variations of features.

SeLeCT is more robust than the DPI-based classifier because layer-4 features are less sensitive to small feature changes than the DPI pattern matching rules. The latter can be fooled by a simple character change.

Fig. 3 gives more insights about the benefits of the filtering steps in the iterative clustering process. It reports the overall accuracy after (i) running the k-means only (line 6 of Alg. 3), (ii) after all clusters with less than $minPoints$ samples have been discarded (lines 4-6 of Alg. 2), and (iii) after the final port based filtering is performed (lines 7-18 of Alg. 2). Accuracy is evaluated at each of the four steps independently of the others, i.e., the results are not cumulative. The first 10,000 flows in the first batch of the Dataset-4S trace are considered. In this case, flows are labelled by the original DPI label; flows in a cluster are then re-assigned the majority label, and the original and the new label are then compared. . Results show that discarding clusters with less than $minPoints$ provides small improvements, while the port-based filtering is the key to boost accuracy to 98% when *dominatedPort* clusters are selected. Only at the last step, when *randomPort* clusters are

considered and the port-based filtering is disabled, accuracy lowers to 82%. In this case, discarding the clusters smaller than $minPoints$ helps improving recall and precision for all classes (see Table III). This last step is important since it allows to properly look for Peer-to-Peer (P2P) protocols that typically do not run on standard server ports.

These results show the benefits of the iterative clustering approach. In particular, they highlight the befits of the filtering mechanisms that allows exploiting the information carried by the $srvPort$, which was not leveraged by previous clustering approaches.

VII. Interesting findings enabled by SeLeCT

One of the interesting possibilities offered by SeLeCT is its ability to automatically group flows in homogeneous clusters. It is thus interesting to verify if the clusters offer more fine-grained classification than traditional protocol classification. We first investigate *dominatedPort* clusters whose DPI inherited label is "Unknown" for all datasets. We found:

• $srvPort = 5223$ - the **Apple push notification server over TLS** is identified in Dataset-3 and Dataset-4;

• $srvPort = 5152$ - **Backdoor.Laphex.Client** traffic is identified in Dataset-1;

• $srvPort = 12350$ - the **Skype proprietary authentication** protocol is identified in Dataset-3 and Dataset-4;

SeLeCT automatically unveils clusters of traffic generated by services that appear as real unknown to the network administrator. This is the case of the *Apple Push Notification* system for iOS devices and iCloud enabled devices, which is based on the SSL/TLS protocol, but running on a non standard $srvPort = 5223$. 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. By running an active experiment, it is possible to confirm that all flows in this cluster are 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. Similarly the cluster of flows directed to $srvPort = 12350$ turns out to unveil *Skype Authentication* protocol traffic. Also in this case, the $srvIP$ of all flows reveals strong clues about the application. All flows are directed to $srvIP$ in the subnet 213.146.189.0/24, registered to Skype Inc.

We then analyze clusters labeled as HTTP traffic. There are several tens of them in each dataset, and some share some clear threat. As proposed in [22], the $srvIP$ feature reveals interesting information. For instance, $srcIP$ addresses in some clusters clearly belong to the same subnet. By means of a simple *whois* query, it is possible to identify clusters containing only Google, Dailymotion or Amazon services, respectively. Similarly, a POP3S cluster refers to *mail.google.com* servers scattered in 4 different subnets in Dataset-4, while a second POP3S cluster aggregates together all flows of other mail providers.

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.

Overall, we were able to find labels for about 90% of unknown clusters. The remaining 10% of clusters contains flows that appear to be encrypted, and for which the IP addresses refer to end-user addresses assigned by ISPs to modems. We suspect those could be Skype flows, but we are not able to confirm this assumption.

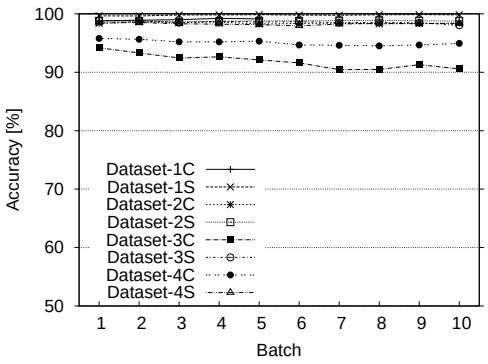


Fig. 4. Accuracy over different batches.

VIII. Exploring the seeding process

So far we have analyzed the performance of SeLeCT considering a single batch provided as input. We are interested now in analyzing the performance of the seeding process. To accomplish this, we run SeLeCT on ten successive batches of flows. As previously done, the bootstrapping at batch 1 is done using the DPI labels. Then, for the subsequent batches, **extractSeeds()** is used to seed the labeling process from batch n to batch $n + 1$. Each batch performance is evaluated by comparing the DPI labels in the ground truth with the labels provided by SeLeCT.

A. Self-seeding

Fig. 4 shows the results for all datasets. First, notice that the accuracy of SeLeCT is extremely high and stable over time for all server datasets. As we already mentioned before, this is due to the better representativeness of the layer-4 features for server flows. Other metrics (i.e., the number of clusters and the percentage of clustered flows) remain unchanged over different batches and hence we do not report these results.

For client Dataset-3C and Dataset-4C, the accuracy slightly decreases over time. For instance, in Dataset-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 higher than 98% for all classes of traffic except for BitTorrent and eMule protocols which tend to be confused with each other. This is detailed by the confusion matrix of the 10-th batch in Table IV. Note that the total number of flows exceeds the batch size, since at step 10 SeLeCT processes also seeding flows. The relative higher fraction of P2P traffic in the Dataset-3C (collected at 2am) results in a

	BT	eMule	HTTP	HTTPS	IMAPS	POP3	POP3S	UNK
BT	**157**	105	0	0	0	0	0	8
eMule	122	**3556**	0	0	0	0	4	24
HTTP	0	0	**10815**	0	0	0	0	5
HTTPS	0	0	1	**1291**	0	0	0	14
IMAPS	0	0	0	0	**53**	0	0	0
POP3	0	0	0	0	0	**145**	0	3
POP3S	0	0	0	0	0	0	**25**	0
UNKNOWN	0	0	18	0	0	0	0	**196**

TABLE IV
CONFUSION MATRIX AT BATCH 10 FOR DATASET-3C.

$SrvPort$	25	80	**88**	110	443	995	**1935**	**4662**	**5223**	**12350**
# cluster	1	46	1	3	30	2	1	51	1	1
Label	SMTP	HTTP	HTTP	POP3	HTTPS	POP3S	RTMP	eMule	Apple	Skype

TABLE V
dominatedPort CLUSTERS AT BATCH 1. BOLD FONT HIGHLIGHTS CLUSTERS ON NON-STANDARD PORTS.

global decrease in the overall accuracy. Similar considerations hold for the Dataset-4C which refers to peak time. However, in this case the fraction of P2P flows is smaller than during the night and thus it has less impact on the overall accuracy. An important and desirable property is that confusion actually happens among P2P protocols only. The lack of dominating port for P2P protocols makes it more challenging for SeLeCT to clearly distinguish the traffic.

Based on the results of our experiments, we believe that SeLeCT shows very good performance in terms of accuracy, precision, and recall. For most protocols, SeLeCT correctly classifies flows for which labels have been provided with no confusion.

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 on these labels for future classification. In the previous experiments we used the labels provided by a DPI to bootstrap the classification and seeding process. 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 Dataset-4S trace and ignore all the DPI labels. In other words, no labels are provided to SeLeCT. At the end of the first batch, the operator has to analyze the clusters that have been formed to label them.

1) dominatedPort Clusters: To assign a label, the information provided by the $srvPort$ for *dominatedPort* clusters proves to be very valuable. Table V reports the $srvPort$ and the number of corresponding *dominatedPort* clusters on the first and second row, respectively, while the third row reports the class label that we assigned. Overall, protocols running on well-known ports are straightforward to identify. Notice that SeLeCT can identify several clusters that refer to the same protocol (e.g., 46 clusters of HTTP flows). In general, the number of clusters is proportional to i) the number of flows, and ii) the variability of the services offered on a given protocol.

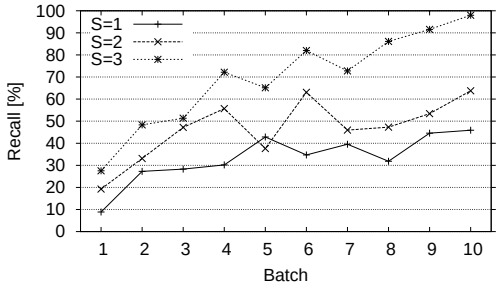


Fig. 5. eMule recall when only S labeled clusters are used as bootstrap at batch 1 for Dataset-4S.

It is interesting that SeLeCT naturally created some clusters whose protocol was not known to the DPI. These clusters are highlighted using bold fonts. By simply searching the web, protocols are easily identified: Port 1935 is used by the Macromedia flash server to stream videos using the RTMP protocol; port 4662 is the default eMule port. At last, port 5223 is used by Apple push notification service for iOS devices running over TLS, and port 12350 cluster contains flows going to Skype Inc. managed servers (see above). Following this approach, 136 clusters can be immediately labeled. Only one cluster dominated by $srvPort = 88$ remains ambiguous. Looking at the closest cluster, it reveals that flows in this cluster are very likely to be HTTP flows, since the 6 closest clusters are HTTP clusters. A simple packet inspection on some flows confirms this hypothesis. This process can be possibly automated in the future.

Once SeLeCT is augmented with the knowledge of these labels, flows are correctly classified in all subsequent batches thanks to the seeding mechanism. From Fig. 7, we can see that more than 80% of flows are typically clustered in *dominatedPort* clusters at the end of step 3. In other words, more than 80% of flows can be easily labeled using simple information obtained from the dominating $srvPort$, whose accuracy is close to 100% (refer to Fig. 3).

2) randomPort clusters: At the last iteration, SeLeCT disables the port filters in **doClustering()** and the remaining 10-20% of flows are clustered in *randomPort* clusters. The analysis of those clusters is expected to be more complicated since the $srvPort$ information is, by construction, providing limited information. First of all, it is easy to see whether a cluster is grouping some P2P protocol or traditional client-server protocols by looking at the $srcIP, dstIP$ of flows, as proposed in [3], [23].

Interestingly, $srvPort$ 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. It is very likely that the port the user would choose is "similar" 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

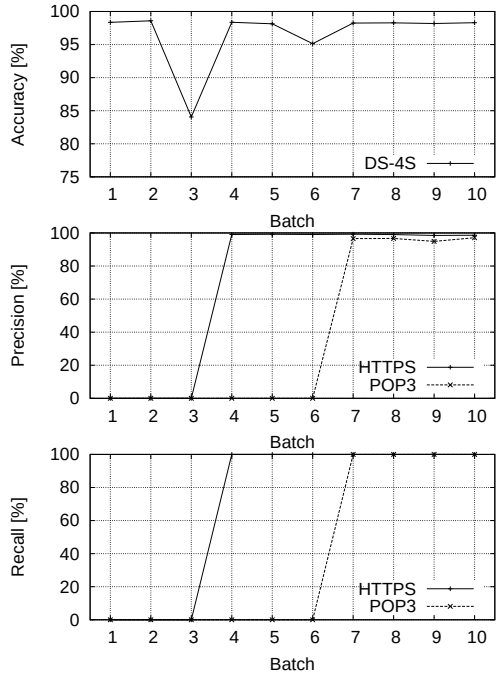


Fig. 6. New protocols suddenly appear: HTTPS traffic is added at batch 3, and POP3 traffic is added at batch 6 in Dataset-4S.

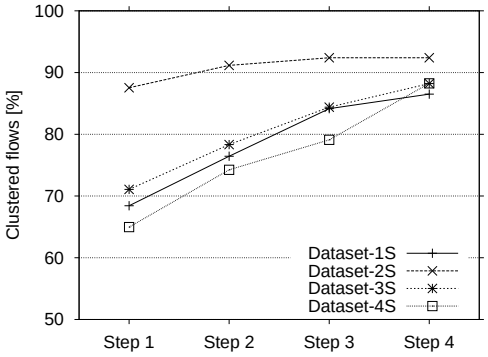


Fig. 7. Fraction of clustered flows at each step.

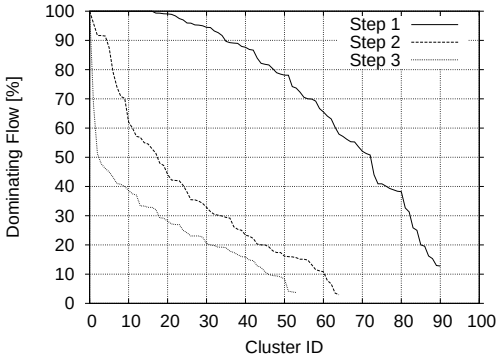


Fig. 8. Fraction of flows directed to the dominating $srvPort$ in each cluster for different steps for Dataset-4S.

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.

At last, packet inspection can been considered as another option to label *randomPort* clusters. Unlike traditional per-flow analysis, the inspection of clustered flows simplifies the identification of signatures since a set of flows is exposed and can be analyzed in parallel to identify common headers. Once a label has been found, SeLeCT extend it to all the flows in the same cluster.

C. Seeding evolution

To show the ability of SeLeCT to increase its knowledge over time, we perform the following experiment. Consider Dataset-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. 5 reports the recall evolution over the different batches for different values of S. For $S = 3$, corresponding to only 28% flows selected as bootstrap at the end of batch 1, SeLeCT already achieves 98% of recall at batch 10. Worst case precision is 98.6%. These results show that SeLeCT seeding process is successfully bootstrapped even if only $S = 1$ cluster is used as initial seed.

We now perform another experiment in which we simulate the sudden appearance of a new class of traffic. We consider the Dataset-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. 6. The top plot reports the overall accuracy, while middle and bottom plots report precision and recall, respectively. Notice how SeLeCT rapidly detects the presence of new traffic classes. In particular, at batch 3, accuracy severely drops since HTTPS flows are labeled as "Unknown". We then bootstrap the HTTPS seeding as before, i.e., by labeling the largest Unknown traffic cluster as HTTPS. Bootstrapping in this case is much faster then for eMule thanks to the purity of HTTPS clusters. Indeed, at batch 4, accuracy returns to 97.5%, and HTTPS precision and recall approach 100%.

At batch 6, the same transient is observed when POP3 flows are injected. Being their number small, the impairment on accuracy is less evident. Then, 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 that SeLeCT allows an easy identification of protocols that, in our example, were not detected by the DPI because no signature was present. This enhances the operator's network visibility by providing homogeneous clusters of flows whose analysis is much easier, due to the aggregated information provided by the flows in the cluster.

IX. Parameter sensitivity analysis

In this section we present an extended set of experiments to evaluate the impact of the parameter choices on SeLeCT. In general, SeLeCT is very robust to various parameter settings

and its behavior is stable in different scenarios. In this section, we report some of the most interesting findings.

A. Setting filtering parameters

Fig. 7 reports the percentage of clustered flows during different iterations of the iterative clustering. Only Server datasets are considered for the sake of simplicity. As we can see, SeLeCT clusters most of the flows during step 1, when there are many *dominatedPort* clusters (i.e., clusters in which most of the flows involve the same port). Small clusters and outlier flows are discarded and passed to step 2. At this point, an additional fraction of *dominatedPort* clusters are identified, allowing to add about 10-15% more flows. This filtering is repeated one more time at step 3 when another 5-10% of flows is clustered. As a last step, SeLeCT looks for *randomPort* clusters and an additional fraction of flows gets properly clustered (e.g., P2P protocols). As the curves suggest, the benefit of adding more *dominatedPort* filtering phases is limited, and little improvement is achieved by setting $itermax$ larger than 3.

To confirm this intuition, Fig. 8 reports, for each step, the fraction of flows directed to the dominating port in each cluster with more than $minPoints$ flows. Clusters are sorted in decreasing fraction for ease of visualization. The number of *dominatedPort* clusters is large during step 1, with 70 clusters having more than 50% of flows that are directed to the same $srvPort$. Given $portFraction = 0.5$, SeLeCT picks flows in these clusters. In step 2, the number of *dominatedPort* clusters decreases, and only 17 clusters pass the $portFraction = 0.5$ filter. In step 3, very few *dominatedPort* clusters are present. This confirms the intuition that it is useless to add more than 3 steps because the information carried by the $srvPort$ has already been exploited. In addition, the intuition suggests to relax the $portFraction$ threshold during the last step, thus we set $portFraction = 0.2$.

B. Sensitivity to $portFraction$

To complete the sensitivity analysis, Fig. 9 shows how the choice of $portFraction$ impacts performance. More specifically, the left plot, which reports the overall accuracy, shows that the impact on accuracy is limited, and only values larger than 80% exhibit some severe degradation on accuracy (note the y-range). The middle plot, which shows the fraction of clustered points, suggests to select smaller values for $portFraction$, since this results in a larger fraction of clustered flows. However, a trade-off is shown in the right plot, because the number of clusters notably increases for small values of $portFraction$. Small values cause the algorithm to accept a lot of clusters in the first filtering steps (refer to Fig. 8), causing the total number of clusters to increase rapidly. Values of $0.3 < portFraction < 0.8$ offer a good trade-off.

C. Sensitivity to k and $minPoints$

Finally, we show the sensitivity of k and $minPoints$ in Fig. 10 and Fig. 11, respectively. Plots report the overall accuracy, number of clusters, and the fraction of clustered flows from left to right, the Client and Server flows on the top and bottom plots, respectively. Fig. 10 shows that accuracy is typically higher than 90% except for very small values of k. Larger values of k improve accuracy, since SeLeCT is allowed to form more clusters. This is confirmed by the total number of clusters which increases almost linearly with k up to a saturation point. However, fragmenting flows into many clusters causes cluster size to be small. Hence, the parameter setting, $minPoints = 20$, filters a larger fraction of flows, causing the percentage of clustered flows to decrease. Finally, notice that Dataset-3C and Dataset-4C are the two most critical scenarios due to the mix of protocols that is present in this network and the relatively weaker descriptiveness of the layer-4 features for client flows.

A similar reasoning applies when varying $minPoints$. It has limited impact on the overall accuracy as already noticed in Fig. 3, while the number of clusters and the fraction of clustered flows exhibit an inverse dependence on $minPoints$: small values cause both of these metrics to grow quickly, while $minPoints$ higher than 15-20 starts showing a saturation. This is true especially for the Server datasets.

Overall, the choice of k and $minPoints$ is not critical; choosing $k = 100$ and $minPoints = 20$ allows a good trade-off between high accuracy, limited number of clusters, and large fraction of clustered flows.

D. Complexity

The complexity of SeLeCT is mainly driven by the complexity of the k-means algorithm. To find the optimal solution considering n objects, k clusters, and a d dimensional space, the problem can be optimally solved in $O(n^{dk+1} \log n)$, which would turn out to be definitively too much for real time applications. However, by considering the centroids computation and re-clustering steps for a fixed number of iterations, the computational time is deterministic. In our case, we choose the number of iteration to be smaller than 1,000,000, and we repeat the k-means 10 times to avoid possible bias do to bad initial centroid choice. Considering these settings, for Dataset-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 even if the current prototype is not optimised. Notice that only flows that have at least 6 data packets are passed to SeLeCT, i.e., 70-90% of flows are actually not considered in practice, see Fig. 1. As a final note, several functions of SeLeCT can also be run in parallel.

X. 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. Given that using unsupervised clustering algorithms does not result in high accuracy, we showed that adding a filtering phase after clustering significantly improves the performance and coverage. Moreover, alternating the clustering and filtering phases further results in very homogeneous clusters, while providing very high coverage.

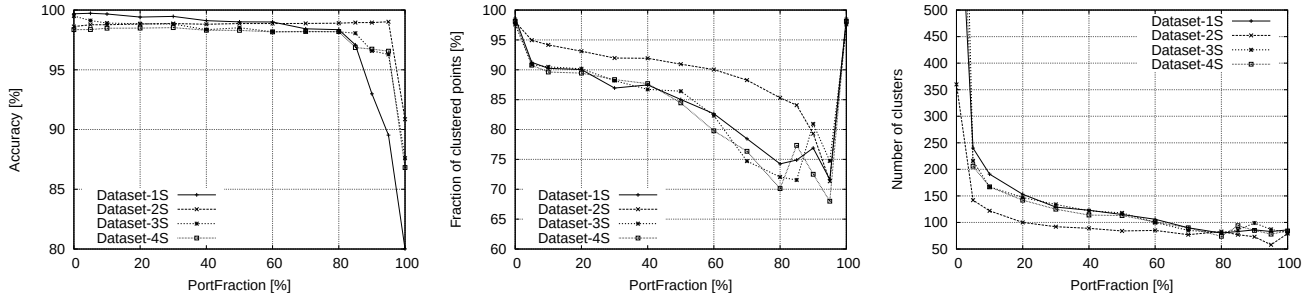


Fig. 9. Sensitivity analysis to $portFraction$: accuracy, fraction of clustered flows and number of clusters in left, middle and right plot.

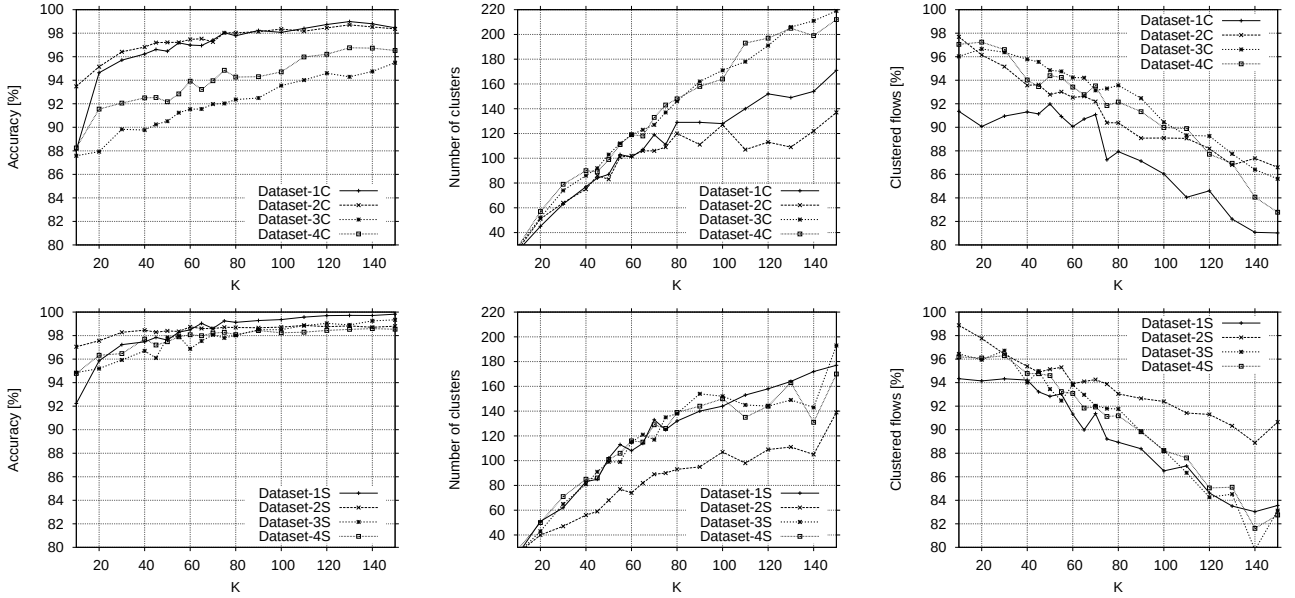


Fig. 10. Sensitivity to k.

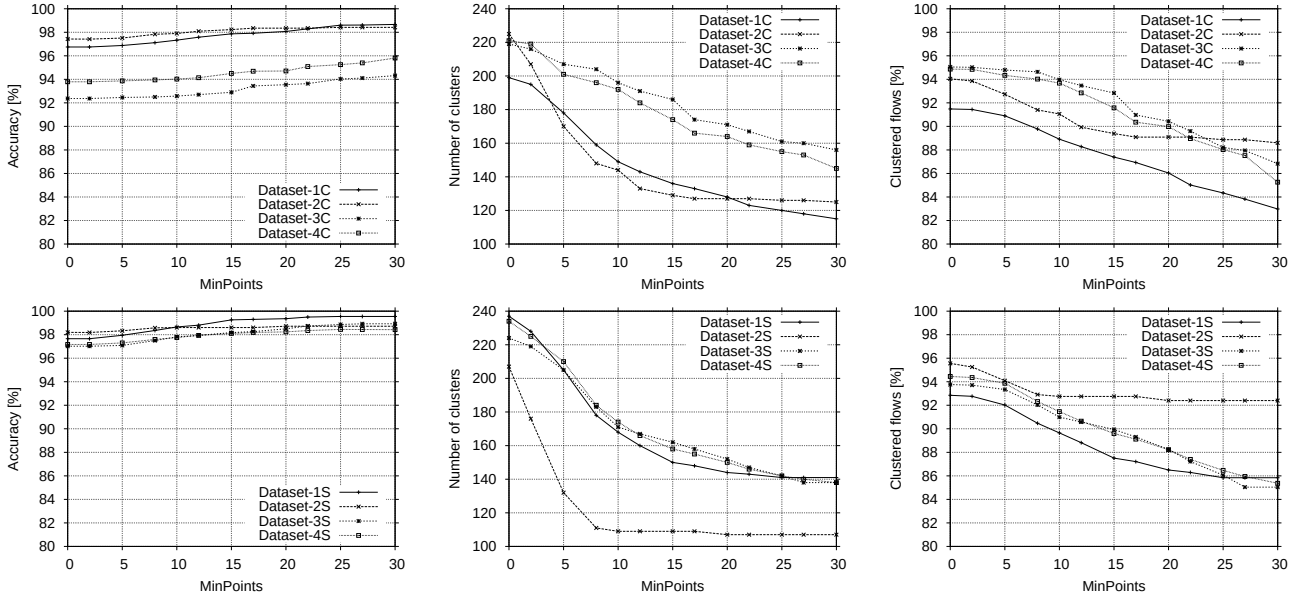


Fig. 11. Sensitivity to $MinPoints$.

Labels for different clusters in SeLeCT can be bootstrapped using several different approaches (DPI, behavioral techniques,

or human-in-the-middle). Once labels for some flows are provided, SeLeCT inherits previously labeled flows to automatically label new clusters. Furthermore, it adapts the model to traffic changes, and is able to automatically increase its knowledge.

Extensive experiments showed that SeLeCT simplifies the manual bootstrapping of labels that, once provided to the system, lead to excellent performance: Accuracy is close to 98% in most datasets, with worst case still higher than 90%. Furthermore, SeLeCT was able to automatically identify classes of traffic that an advanced DPI-based classifier was ignoring like, e.g., the Apple iOS push notification protocol, or some Bot/Trojan traffic.

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, December 2009.

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

[4] 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, October 2007.

[5] L.Grimaudo, M.Mellia, E.Baralis, R.Keralapura, "Self-Learning Classifier for Internet Traffic", *The 5th IEEE International Traffic Monitoring and Analysis Workshop (TMA 2013)*, Turin, IT, April 19th 2013.

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

[7] X. Zhu. Semi-Supervised Learning Literature Survey. *Computer Sciences, University of Wisconsin-Madison*, TR 1530, 2005.

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

[9] 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, May 2002.

[10] A. McGregor, M. Hall, P. Lorier, J. Brunskill. Flow clustering using machine learning techniques. *PAM 2004*, Antibes Juan-les-Pins, FR, April 2004.

[11] J. Erman, A. Mahanti, M. Arlitt. Internet traffic identification using machine learning. *IEEE GLOBECOM*, San Francisco, CA, December 2006.

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

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

[14] 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.

[15] J. Yuan, Z. Li, R. Yuan. Information entropy based clustering method for unsupervised internet traffic classification. *IEEE ICC*, Beijing, CN, May 2008.

[16] P. Casas, J. Mazel, P. Owezarski, MINETRAC: Mining flows for unsupervised analysis & semi-supervised classification, *Teletraffic Congress (ITC), 23rd International*, San Francisco, CA, September 2011.

[17] J. Zhang, C Chen, Y. Xiang, W. Zhou, A.V. Vasilakos, An Effective Network Traffic Classification Method with Unknown Flow Detection, *N*etwork and Service Management, IEEE Transactions on, v.10, n.2, pp.133-147, June 2013.

[18] J. Zhang, Y. Xiang, W. Zhou, Y. Wang, Unsupervised traffic classification using flow statistical properties and IP packet payload, *Journal of Computer and System Sciences*, Volume 79, Issue 5, pp.573-585, August 2013.

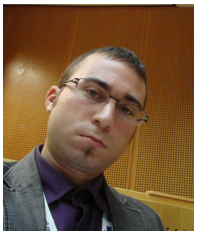
[19] G.Iannaccone, C.Diot, I.Graham, N.McKeown, Monitoring very high speed links. *st ACM SIGCOMM Workshop on Internet Measurement (IMW '01)*. New York, NY, USA.

[20] 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, April 2010.

[21] A.Finamore, M.Mellia, M.Meo, M.Munafo, D.Rossi, "Experiences of Internet traffic monitoring with tstat," Network, IEEE, vol.25, no.3, pp.8-14, May-June 2011.

[22] P. Casas, P. Fiadino, A. Bar, IP Mining: Extracting Knowledge from the Dynamics of the Internet Addressing Space, *Teletraffic Congress (ITC), 25th International*, Shanghai, CN, September 2013.

[23] M.Iliofotou, M. Faloutsos, M. Mitzenmacher. Exploiting Dynamicity in Graph-based Traffic Analysis: Techniques and Applications. *ACM CoNEXT*, Rome, IT, December 2009.



Luigi Grimaudo received the B.S. Degree in Computer Engineering from the Università Degli Studi Di Palermo, Italy, in 2008 and the M.S. Degree in Computer Engineering from the Politecnico di Torino, Italy, in 2010. He is currently pursuing a Ph.D. Degree in Information and System Engineering at Politecnico di Torino, Italy. Since 2011 he is collaborating with Narus, Inc. working on traffic classification problems and on-line social networks analysis. His research interests cover the areas of internet traffic classification, recommendation system, social network analysis and big data.

Marco Mellia graduated from the Politecnico di Torino with Ph.D. in Electronic and Telecommunication Engineering in 2001. He has co-authored over 200 papers published in international journals and presented in leading international conferences, all of them in the area of telecommunication networks. He participated in the program committees of several conferences including ACM SIGCOMM, ACM CoNEXT, IEEE Infocom, IEEE Globecom and IEEE ICC, and he is Area Editor of ACM CCR. His research interest are in the design of energy efficient networks (green networks) and in traffic monitoring and analysis.

Elena Baralis has been a full professor at the Dipartimento di Automatica e Informatica of the Politecnico di Torino since January 2005. She holds a Master degree in Electrical Engineering and a Ph.D. in Computer Engineering, both from Politecnico di Torino. Her current research interests are in the field of database systems and data mining, more specifically on mining algorithms for very large databases and sensor/stream data analysis. She has published over 80 papers in international journals and conference proceedings. She has served on the program committees or as area chair of several international conferences and workshops, among which VLDB, IEEE ICDM, ACM SAC, DaWak, ACM CIKM, PKDD.

Ram Keralapura is currently a Principal Data Scientist at Netskope. He got his BE from Bangalore University and his MS/PhD from University of California, Davis. His dissertation broadly focused on Network Management in ISP networks and in particular dealt with modeling routing dynamics in IGP and BGP protocols. Prior to Netkkope, Ram worked at Narus as a Principal Member of Technical Staff where he led several innovative projects in the Office of CTO. He has published over 30 conference/journal articles and also has over 20 patent applications. His research interests are in cloud apps, distributed networks, Internet routing, traffic engineering and security.