Probabilistic Graphical Models for Semi-Supervised Traffic Classification *

Charalampos Rotsos
Computer Laboratory,
University of Cambridge
cr409@cam.ac.uk

Jurgen Van Gael
Engeneering Department,
University of Cambridge
jv279@cam.ac.uk

Andrew W. Moore
Computer Laboratory,
University of Cambridge
awm22@cam.ac.uk

Zoubin Ghahramani
Engineering Department,
University of Cambridge
zoubin@eng.cam.ac.uk

ABSTRACT

Traffic classification using machine learning continues to be an active research area. The majority of work in this area uses *off-the-shelf* machine learning tools and treats them as *black-box* classifiers. This approach turns all the modelling complexity into a feature selection problem. In this paper, we build a problem-specific solution to the traffic classification problem by designing a custom probabilistic graphical model. Graphical models are a modular framework to design classifiers which incorporate domain-specific knowledge. More specifically, our solution introduces semi-supervised learning which means we learn from both labelled and unlabelled traffic flows. We show that our solution performs competitively compared to previous approaches while using less data and simpler features.

Categories and Subject Descriptors

C.2.3 [**Computer Systems Organization**]: COMPUTER-COMMUNICATION NETWORKS Network Operations[Network monitoring]

General Terms

Measurements, Algorithms

Keywords

Traffic classification, Semi-supervised learning, probabilistic graphical models

1. INTRODUCTION

Application identification is the task of identifying the type of application that generates a particular network flow. With the increase in the complexity of computer networks and explosion of the number of network applications, application identification has become an increasingly important problem for the research community. Lightweight and accurate application identification methods can make resource allocation more effective, since this variability in applications creates conflicting resource requirements that are difficult to fulfill. Additionally, the increase in the diversity of applications makes network tasks like administration, network planning and policing difficult.

For many years, the main method to infer the application of a network flow was the port number of the server. This approach requires that hosts use specifically assigned ports for each application. Research shows that simple port-based application identification have poor results for some applications, like Skype and p2p-filesharing [11].

In order to overcome the deficiency of port-based methods, complex application-identification frameworks have been discussed in the literature that try to incorporate extended information sources from the network in order to increase their accuracy [8, 12]. The majority of such work uses packet level information, requiring e.g. expensive hardware for network measurements. Additionally, most of the methods require both expensive manual tagging of training datasets and extensive monitoring of applications for protocol changes.

In this work, we use flow record data (*e.g.* NetFlow) for application identification. ISPs have flow-level measurement infrastructure in place for billing and anomaly detection purposes. Hence, this type of data makes for an appealing information source to build our application identification framework. Additionally, previous work on the problem has shown that simple machine learning algorithms can achieve good identification accuracies [7].

We present a solution based on probabilistic graphical mod-

els [9]. Probabilistic graphical models have a number of advantages over off-the-shel classifiers such as SVM's, decision trees, etc. a) they are modular and hence can be re-used and easily extended, b) their visual representation helps clarifying the model assumptions, and c) all necessary computations involved in the analysis can be expressed as simple manipulations of graphs.

In this work we build a semi-supervised learning algorithm using probabilistic graphical models. Where traditional classifiers learn *only* from labelled data, a semi-supervised classifier learns from both labelled and unlabelled data. This is particularly relevant for application identification as it is easy to gather large sets of flows but expensive to label them. More specifically, we extend the Naive Bayes classifier by allowing it to incorporate unlabelled data. We introduce and compare two algorithms to learn our semi-supervised classifier. The learning algorithms differ by how they integrate unlabelled data. As far as we know, this is the first attempt to extend traditional classifiers using the framework of probabilistic graphical models in the field of application identification.

2. METHODOLOGY

Classification is the process by which one is given a set of training examples for various different classes and the goal is to build an algorithm which classifies previously unseen datapoints.

When training a classifier from data, we can choose between a *supervised learning* approach or an *unsupervised learning* approach. In the former, we are given a labelled dataset to train the classifier. Although establishing ground truth is expensive, the labels provide valuable information. In unsupervised learning setting, learning purely from data, the lack of labels means that gathering the data is cheap but the classifier might be much less accurate.

In our setting, we ask the question whether we can use unlabelled network flows to improve the accuracy of a classifier that is trained on a small set of labelled network flows. This approach is commonly known as *semi-supervised learning* [15].

There are many powerful semi-supervised learning methods available; we chose a method based on the following two criteria:

- the method needs to be flexible enough so we can encode domain specific knowledge, and
- the method needs to scale to datasets with potentially millions of datapoints.

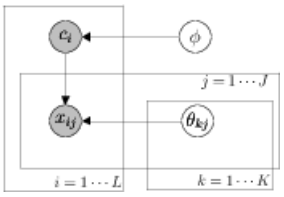
The first point is crucial: initial experiments with a number of black box machine learning algorithms work well as a baseline but are soon limited, being hard to improve with domain expert knowledge. *E.g.*, the IP address is treated as a sequential number rather than incorporating its subnetwork structure. The second point is a practical consideration which excludes many advanced machine learning models; *e.g.*, graph based semi-supervised learning is a very powerful technique, but we have yet to find algorithms that easily scale to hundreds of thousands of datapoints.

We chose to work within the framework of probabilistic graphical models [9][1]. This offers some unique advantages:

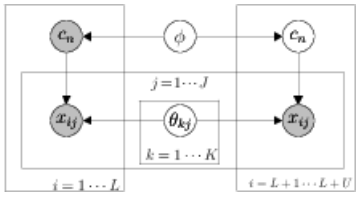
- *modularity*: it is easy to replace part of the graphical model with a more sophisticated version that adapts better to the data,
- *iterative design*: if we can identify the source of model mis-specification, we can iteratively refine a graphical model, and
- *unifying framework*: many machine learning models can be cast into the language of graphical models, hence we can build on the growing understanding and expertisve of other approaches.

As we describe shortly, semi-supervised learning fits easily into the framework. We do not have to change the model for a lack of labels, only the algorithms used to infer labels from the model.

Before we describe our particular model, we formalize the classification task. We are given a set of datapoints $X = \{x_1, x_2, \cdots, x_L\}$ where x_i represents network flow i, and a set of corresponding class labels $C_X = \{c_1, c_2, \cdots, c_L\}$ where each c_i represents a particular application. Moreover, we assume there is a set $Y = \{x_{L+1}, \cdots, x_{L+U}\}$ of unlabelled network flows. Each datapoint x_i consists of various measurements with x_{ij} denoting the jth feature of flow i. Our goal is to train a classifier $f(\cdot)$ using the sets X, Y, C_X such that for any network flow x^* we can predict a good class label $f(x^*)$. Using the graphical model formalism we will do this in two steps: first we define a probability distribution over all known and unknown variables in our application and, second, we specify a rule which says how the probability distribution relates to our decision rule f. The starting



(a) supervised classifier



(b) semi-supervised classifier

Figure 1: The graphical model of the classifier

[1]We refer to *probabilistic graphical models* and *graphical models* interchangeably throughout the remaineder of this paper.

point for our semi-supervised approach is the following generative process for the data: there is a random variable c_i which denotes the class label for flow i. This random variable is drawn from a distribution over classes $p(c_i|\phi)$ where ϕ parametrizes the distribution over classes. Generally, we choose $p(c_i|\phi)$ to be a discrete distribution with probability vector ϕ. In other words, $p(c_i = a|\phi) = \phi_a$. Once the variable c_i is drawn, we draw each feature j for datapoint i independently from a distribution $p(x_{ij}|\theta, c_i)$. We choose $p(x_{ij}|\theta, c_i)$ to be a discrete distribution with probability vector $\theta_{c_i,j}$. In the case where x_{ij} is a continuous quantity, we divide the range of x_{ij} into buckets (perhaps after a log transformation) and each bucket is now a discrete value. This assumption implies that we are learning a histogram for each continuous quantity. Figure 1(a) illustrates the graphical model for our classifier. This type of model is commonly known as the *Naive Bayes* model.

The main assumption of the Naive Bayes model is the independence of the features. This is certainly not the case in our network flow context: *e.g.* IP addresses and ports are strongly correlated. Nonetheless, this assumption works sufficiently well, provides mathematical tractability and permits a comparison baseline.

To summarize the model so far, we have defined a joint probability distribution over the class labels and features $p(c_i, x_i|\phi, \theta) = p(c_i|\phi)\prod_j p(x_{ij}|\theta, c_i)$. Moreover, we assume that all flow datapoints are generated independently and identically distributed, according to this distribution. In other words, the joint distribution over all datapoints and labels is

$$p(C_X, X, C_Y, Y|\phi, \theta) = \prod_{i=1}^{L+U}\left(p(c_i|\phi)\prod_j p(x_{ij}|\theta, c_i)\right). \quad (1)$$

We do not know the values for ϕ and θ but we can learn them from data. Although there are various ways to do so, we will take a Bayesian approach, introduce prior distributions on these parameters and average over their possible values. Since both random variables ϕ and θ represent probability vectors, we choose a distribution over probability vectors called the symmetric Dirichlet distribution: $\phi \sim \text{Dirichlet}(\alpha_\phi)$ and $\theta_{cj} \sim \text{Dirichlet}(\alpha_\theta)$ for each class label c and feature j. The symmetric Dirichlet distribution is the most common distribution over probability vectors and has the property that Dirichlet(α) will tend to be close to uniform when α is large and sparse when α is small. In other words, α decides whether all or only a few components of the drawn probability vector get probability mass. For our application, a small α_ϕ encodes our belief that few classes will get all the probability mass. This could, for example, encode that on a particular network, we mostly see web traffic.

Our end goal is to find good settings of the parameter ϕ and θ such that we can use them to predict the label of a new datapoint. The Bayesian approach makes this particularly elegant. First we learn the *posterior distribution* over our parameter ϕ and θ conditional on the data: $p(\phi, \theta|X, Y, C_X)$. This distribution captures our belief about the parameter of our Naive Bayes model after processing the data. Then, we use the posterior distribution over ϕ and θ to predict the class label of a new network flow by applying Bayes' rule. Given a new network flow x^*, we can compute $p(c^*|x^*, X, Y, C_X)$ by integrating over the posterior distribution of parameters $p(c^*|x^*, X, Y, C_X) \propto \int p(c^*|\phi)p(x^*|\theta, c^*)p(\phi, \theta|X, Y, C_X)d\phi d\theta$. The distribution $p(c^*|x^*)$ captures everything we know from our labelled and unlabelled data.

Finally, for the purpose of this paper we classify x^* in the class with the highest probability $f(x^*) = max_c(p(c|x^*))$. Note that one can easily apply a more general decision theoretic approach by introducing different mis-classification cost for different classes.

Naive Bayes Training

In a completely supervised setting with no datapoints in Y, C_Y, (left plot in Figure 1(a)) computing the posterior $p(\phi, \theta|X, Y, C_X)$ reduces to a simple counting problem. We leave the derivation out for brevity and just describe the posterior computations: the posterior of ϕ is Dirichlet($\alpha_{\theta,1} + n_1, \cdots, \alpha_{\theta,N} + n_N$) where $\alpha_{\theta,i}$ is the prior for class i and n_i is the number of datapoints in class i. Similarly, the posterior for θ_{ij} is a Dirichlet distribution where the parameter is the sum of the prior and the number of occurrences of feature j in class i.

In our setting, by introducing unlabelled datapoints Y, the computation of the posterior over our parameters $p(\phi, \theta|X, Y, C_X)$ is intractable. More specifically, using Bayes' rule we find that

$$p(\phi, \theta|X, Y, C_X) \propto \sum_{C_Y} p(C_X, X, C_Y, Y|\phi, \theta)p(\phi)p(\theta). \quad (2)$$

This equation requires computing all possible label assignments for the unlabelled flows in Y. A brute force way of computing this exponentially large sum is prohibitively expensive. In the following paragraphs we introduce two algorithms which approximate this sum.

The first algorithm, The *hard assignment* variant is an algorithm in which we compute the posterior distribution based on the labelled data, $p(\phi, \theta|X, C_X)$, and we iteratively add one unlabelled datapoint as follows: for each $x^* \in X$ we compute the posterior class label $p(c^*|x^*, X, C_X)$. We then assign x^* to the class with the highest probability mass and include it in our labelled set X and C_X while recomputing the posterior $p(\phi, \theta|X, C_X)$. This algorithm is easy to implement, has the same computational overhead as the supervised Naive Bayes classifier, but unfortunately ignores the uncertainty in class label c^*.

The second algorithm which we tested, the *soft assignment* variant, takes the uncertainty of c^* into account. The algorithm is exactly the same as the hard assignment algorithm, except that when we include one new datapoint, we update the posterior for each parameter according to the predicted weight of the datapoint. *E.g.* if a datapoint is in class 1 with probability 0.3 and in class 2 with probability 0.7, we will partially update θ_1 and partially update θ_2. This approach has a solid theoretical foundation and is commonly known as the Online Expectation Maximization algorithm [10].

3. RESULTS

3.1 Experimental Data

Table 1: Composition of dataset

app	%	app	%	app	%
DB (mysql)	0.0430	SERV (time)	0.0003	P2P (file-share)	0.1147
MAIL	0.0250	SPAM	0.0048	WEB	0.7233
FTP	0.0625	STREAM (rtp)	0.0031	VPN	0.001
IM	0.0060	VOIP	0.0016	ACCESS (ssh)	0.0061

Table 2: Features of flow records.

features	Explanations
srcIP/dstIP	source/destination IP address
srcPort/dstPort	source/destination port address
tos	IP type of service
sTime/eTime	flow start/end timestamp
tcpFlag	cumulative OR of TCP flags
bytes	total number of bytes observed
pkts	number of packets observed
length	duration of flow (eTimeA|sTime)
pktSize	average packet size (bytes/pkts)
byteRate	average flow rate (bytes/length)
pktRate	average packet rate (pkts/length)
tcpFxxx	xxx=syn/ack/fin/rst/psh/urg flag

For the evaluation of the proposed methods, we use a single network trace from a large multi thousand user research institute. The trace covers two consecutive weekdays in 2006. For the labeling process we used GTVS[1] on the full payloaded trace. Table 3 shows the frequencies of application labels in the trace.

For the extraction of the flow records, we used nProbe [3], a software implementation of a NetFlow record extractor. The resulting dataset contains approximately 6 million NetFlow records. For the extraction, we use the configuration for a Tier-1 ISP network and no packet sampling was applied.

3.2 Experiments

We implemented both the **hard** and **soft** semi-supervised classifiers in C#, which, for brevity, we call **hard** and **soft** respectively. In addition, we used the Naive Bayes implementation from the Weka framework, with Kernel density estimation for modeling the numeric features [6], in order to build **NBK**, a model for baseline comparison. We call the last classifier **NBK**. Table 2 shows the feature list that we used in order to represent every datapoint. In order to address the capabilities of our approach we consider: 1) How good is our approach in terms of accuracy? 2) How efficient is our approach for small training datasets? and 3) How does our model perform when we vary model parameters?.

3.2.1 *Baseline model comparison*

Figure 2 shows the accuracy of the different approaches over the full trace. All the classifiers which are presented have been trained over the first 30 minutes of the trace and the resulting model is used to classify the rest of the trace. For the computation of the accuracy we segment the trace into non-overlapping slices of 30 minutes and plot the accuracy over each subtrace.

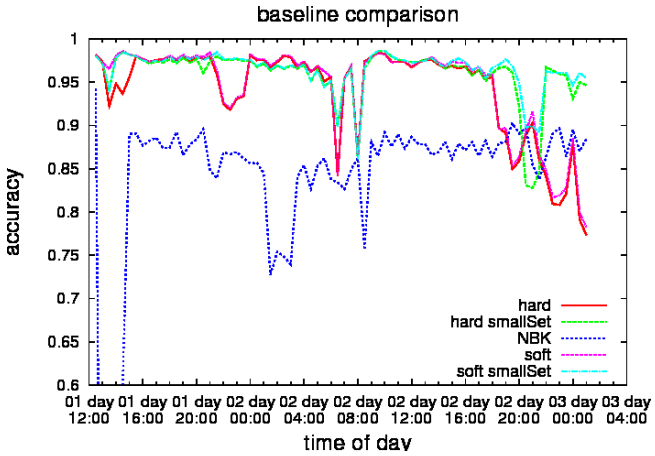


Figure 2: Comparison of classifier using the same training dataset

In Figure 2 we can see that the **NBK** classifier is constantly less in accuracy than all other classifiers. The low accuracy of the **NBK** model becomes more apparent during out of hours, where its performance has higher variance. This is a result of the changes in observed application mix. Because the **NBK** model has low accuracy on some classes, the overall performance for times 00:00 to 4:00 in Day 1 is below 85%. At the same time, the effect of these changes is reduced in the accuracy of the other classifiers. In order for the **NBK** model to be optimized, we used kernel estimation for the modeling of the numeric features. This modeling approach has been applied in previous work [7] and proves to be highly effective. The kernel estimation approach models the various features of the trace as an infinite sum of Gaussians. We predict that the kernel estimation is probably overfitting the model on the data. While this approach can be beneficial to some features like duration, it results in bad modeling of features like IP addresses and ports, because the use of a specific port does not imply the usage of near-by ports from the same application. Without kernel estimation, the Naive Bayes model assumes that a numeric value is generated by a single Gaussian process and this results in an even lower performance of the classifier.

In Table 3, we provide a per-class accuracy of the different classifiers we compare. We observe that the majority of the classes have only small differences between the different semi-supervised algorithms. Although some applications have better performance with one of the two algorithms; this can partially be explained because of the way that the two algorithms perform their updates. The **hard** approach performs better for *STREAM* and *NEWS*. These applications have more static features (port number, well known hosts *etc.*) and the **hard** assignment increases faster the certainty over these values. The **soft** approach performs better for the classes of *VPN* and *IM*. These classes are less common and have high variance over the feature values.

Table 3: Average accuracy for each application class

	DB	MAIL	FTP	IM	P2P	ACCESS
Hard	1	0.58	1	0.39	1	0.95
Hard-ss	1	0.59	1	0.82	1	0.77
Soft	1	0.55	1	0.42	1	0.96
Soft-ss	1	0.61	1	0.42	1	0.81
NBK	0.84	0.26	0.42	0.76	0.91	0.11
	SERV	SPAM	STREAM	WEB	VPN	VOIP
Hard	0	1	0.97	0.99	0.82	0.24
Hard-ss	0	1	0.91	0.99	0	0.44
Soft	0	1	0.96	0.99	1	0.77
Soft-ss	0	1	0.96	0.99	0.03	0.21
NBK	0.24	0.95	0.1	0.89	0.35	0.12

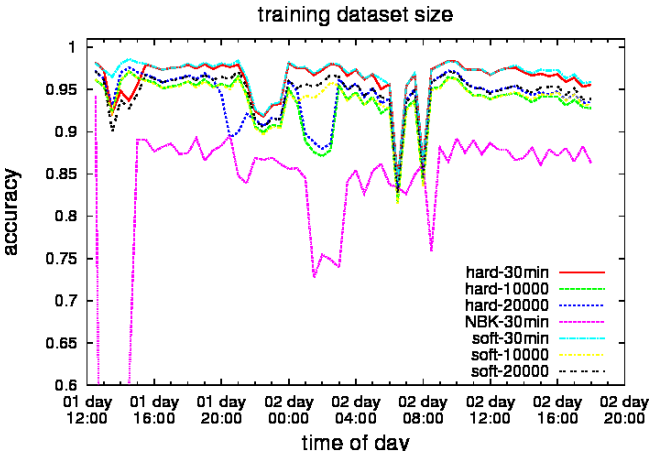


Figure 3: Comparison of classifiers using variable size training dataset

Figure 2 illustrates how inaccurate the **hard** and **soft** assignment algorithms are near the end of the trace. The intuition behind this performance degradation is that the classifier doesn't readily adapts to new datapoints as the training dataset grows. For this reason we develop a complementary model for each algorithm which uses a reduced training dataset which are called **hard-ss** and **small-ss** respectively. The reduced dataset contains the initial half hour labelled subtrace and the latest half hour of data, labelled by the classifier. The resulting models can be seen in Figure 2. The ss (small set) models seems to overcome the problem of accuracy reduction near the end of the trace, but such an approach is inefficient because it discards important information. We are although currently studying ways that could compress the initial dataset so that the model can adjust to new datapoints after some time without any information loss.

3.2.2 *Training dataset sizes*

One of the main qualities of the semi-supervised approach, as it is described in Section 2, is its ability to achieve high accuracy when trained on a relatively small labelled dataset. In order to test this we trained the semi-supervised algorithms with a reduced training dataset and compared the

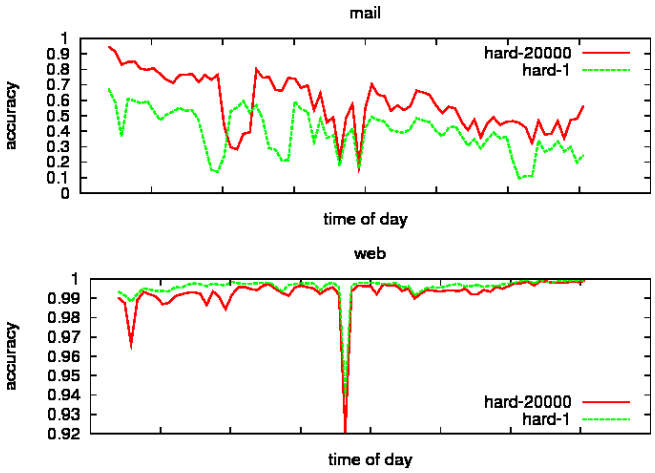


Figure 4: Comparison of classifier with different Dirichlet parameters

accuracies over time, shown in Figure 3. We used three training datasets to build a model for each semi-supervised algorithm: the first half hour of the trace (which contains approximately 130000 flow records), 10000 and 20000 randomly selected flows. The flows that are included in the training process are excluded from the classification process. We also include the results of the **NBK** model trained on the first 30 minutes of the trace.

As we can see in Figure 3, the semi-supervised algorithms achieve high accuracy — above 90% most of the time — regardless of the size of the prior knowledge. The classifiers trained with smaller datasets have a small degradation on their performance of around 1-5%, but they compensate by requiring only a small amount of labelled data. The resulting models also manage to outperform the **NBK** model throughout the whole trace, even though the **NBK** model uses a larger training dataset.

3.2.3 *Parametrization of the model*

As we discussed in Section 2, the Bayesian approach which we employ allows easy adaptation of the model to different environments by providing a simple mechanism for the expression of prior knowledge that we might have from the problem domain. This can be achieved by controlling the type and the parameters of the distributions of the priors for the random variables of the problem. In Figure 4, we plot the accuracy of two **hard** assignment classifiers over time in a way similar to the previous experiments, but only for the classes of *WEB* and *MAIL*. The difference between the two models is that the **hard-20000** model has a Dirichlet prior for the parameter of the class membership variable with parameter $\alpha_\theta = 20000$, while the **hard-1** model has a Dirichlet prior with parameter $\alpha_\theta = 1$. The parameter of the Dirichlet distribution α_ϕ for the parameter of the feature distribution of each class is the same for both models and equal to 1.

The choice of high α_θ parameters for the **hard-20000** model expresses our assumption of uniformity with high confidence for the distribution of the labels. Variation in the label frequencies in the training mix will have small impact in the posterior distribution of the label distribution, and frequent

labels will be less favoured in the classification process. As a result, the *WEB* class has a slightly lower accuracy in the **hard-20000** model, but the same model has better performance for the *MAIL* class, which is less frequent. From this small experiment, it can be seen that the Bayesian approach to data modeling can be a powerful tool when we know some of the statistical properties of the data we are classifying, or when we want to exhibit specific capabilities from our model.

4. RELATED WORK

Although probabilistic graphical models are quite common in language processing and bio-informatics, they have not yet been explicitly applied for application identification problems. A series of papers has been written covering a wide range of other machine learning techniques, both using the supervised and unsupervised paradigm [12, 8]. Most of this work is focused on modeling packet level information. An early literature review on the specific subject can be found in [13]. The work most relevant to our approach can be found in [5], in which the authors present a semi-supervised approach to the problem of traffic classification using a *K-means clustering* algorithm on both labelled and unlabelled data that casts a majority vote in order to define the label of a cluster.

The first paper that discussed the problem of traffic classification using flow records was [7], a Naive Bayes classifier with kernel estimation is used on labelled data, and reports very high accuracy. The paper concludes that the accuracy of a Naive Bayes model can increase when it is combined with smart feature engineering even on sampled data. An extension of this approach can be found in [2]. The author build a C4.5 classifier over NetFlow data and report good results. In addition they discuss problems that may occur when data are sampled.

5. CONCLUSIONS AND FUTURE WORK

In this paper we present a semi-supervised approach to a NetFlow-based traffic classification methodology. We explore two algorithms for training the semi-supervised classifier. The model described exports a set of well defined parameters that allow, unlike methods like SVM, the easier adaptation of the model to the requirements of the classification process and achieves very good results with a significantly reduced training dataset. We intend this to be our first step towards a system for traffic classification focused on information poor data-sources.

Although our results are encouraging, we see two main limitations. First of all, we don't model the effects of packet sampling. We know from theory that random sampling would increase variance in measurements [4] and make discrimination more difficult. Secondly, we are aware that our model might behave badly when testing and training environment are different, because a portion of its accuracy depends on the stability of the IP address.

Luckily the probabilistic graphical model approach is flexible and we are currently working to extend our solution by incorporating more of our domain structure. E.g. we are currently trying to fuse the initial information with extra fields by aggregating flows on different levels and incorporating in the existing model collective inference [14] techniques that exploit the connection patterns of the hosts.

6. REFERENCES

[1] Canini, M. *et al.*. GTVS: Boosting the collection of application traffic ground truth. In *TMA '09: Proc. of the First International Workshop on Traffic Monitoring and Analysis*. Springer-Verlag.

[2] Carela-Espanol, V. *et al.*. Traffic classification with sampled netflow. Technical report, Universitat Politecnica de Catalunya, 2009.

[3] L. Deri. nProbe: an open source netflow probe for gigabit networks. In *In Proc. of Terena TNC 2003*.

[4] Duffield, N. *et al.*. Estimating flow distributions from sampled flow statistics. In *SIGCOMM '03: Proc. of the 2003 conf. on Applications, technologies, architectures, and protocols for computer communications*. ACM.

[5] Erman, J. *et al.*. Semi-supervised network traffic classification. In *SIGMETRICS '07: Proc. of the 2007 ACM SIGMETRICS international conf. on Measurement and modeling of computer systems*. ACM.

[6] Hall, M. *et al.*. The WEKA data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1), 2009.

[7] Jiang, H. *et al.*. Lightweight application classification for network management. In *INM '07: Proc. of the 2007 SIGCOMM workshop on Internet network management*. ACM.

[8] Kim, H. *et al.*. Internet traffic classification demystified: myths, caveats, and the best practices. In *CONEXT '08: Proc. of the 2008 ACM CoNEXT Conf.* ACM, 2008.

[9] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[10] P. Liang and D. Klein. Online em for unsupervised models. In *NAACL '09: Proc. of Human Language Technologies: The 2009 Annual Conf. of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics.

[11] Maier, Gr. *et al.*. On dominant characteristics of residential broadband internet traffic. In *IMC '09: Proc. of the 9th ACM SIGCOMM conf. on Internet measurement conference*. ACM.

[12] A. W. Moore and D. Zuev. Internet traffic classification using bayesian analysis techniques. In *SIGMETRICS '05: Proc. of the 2005 ACM SIGMETRICS international conf. on Measurement and modeling of computer systems*. ACM.

[13] T. Nguyen and G. Armitage. A survey of techniques for internet traffic classification using machine learning. *Communications Surveys and Tutorials, IEEE*, 10(4), Quarter 2008.

[14] Sen, P. *et al.*. Collective classification in network data. *AI Magazine*, 29(3), 2008.

[15] X. Zhu. Semi-Supervised learning literature survey. Technical Report 1530, Computer Sciences, University of Wisconsin-Madison, 2005.