Internet Archeology: Estimating Individual Application Trends in Incomplete Historic Traffic Traces

Sebastian Zander, Nigel Williams, Grenville Armitage
Centre for Advanced Internet Architectures. Technical Report 060313A
Swinburne University of Technology
Melbourne, Australia
{szander, niwilliams, garmitage}@swin.edu.au

Abstract--**Public traffic traces are often obfuscated for privacy reasons, leaving network historians with only port numbers from which to identify past application traffic trends. However, it is misleading to make assumptions based on default port numbers for many applications (such as peer-to-peer file sharing or online games). Traffic classification based on machine learning could provide a solution. By training a classifier using representative traffic samples, we can classify (and differentiate between) distinct, but possibly similar, applications of interest in previously anonymised trace files. Using popular peer-to-peer and online game applications as examples, we show that their traffic flows can be separated after-the-fact without using port numbers or packet payload. We also address how to obtain negative training examples, propose an approach that works with any existing supervised machine-learning algorithm, and present a preliminary evaluation based on real traffic data.**

Keywords--**Traffic Classification, Machine Learning**

I. Introduction

Over the last decade a large number of traffic traces have been captured within the Internet. Estimating traffic trends in these traces is essential in uncovering past network usage and could help predicting future usage. Currently the most reliable method of identifying applications is through packet payload inspection. This approach is limited however, as often payload data is either partially represented (few bytes of the payload are captured) or absent altogether. Even in cases where a few bytes of payload remain, studies have shown accurate identification to be difficult [1]. This leaves port numbers as only method of estimating traffic trends.

With an ever-increasing number of network applications, extensive use of network address translation (NAT), dynamic port allocation and end-users deliberately choosing non-default ports, port-based identification is becoming ineffectual. The most prominent examples are peer-to-peer file sharing (p2p) applications, of which a significant amount of traffic is found on non-default ports (see [2]).

Machine learning (ML) is a possible solution to overcome the shortcomings of port and payload-based analysis. ML algorithms can be trained on data that describes each application by packet payload independent traffic characteristics (e.g. packet length, inter-arrival time distributions), and as such are ideal for use with anonymised traces. Previous work has predominantly focussed on the classification of flows into application types, such as interactive and non-interactive. Our approach differs in that we attempt to classify flows into specific applications, which presents several new challenges. Firstly, we need to differentiate between distinct applications with possibly similar traffic profiles. Secondly, we need to obtain a proper set of data to train the ML algorithm. This dataset must include representative data for each application of interest (positive examples) and data representing all other applications (negative examples).

Obtaining the positive examples 'only' requires some representative traffic of the applications of interest. That traffic could be captured in the network or traffic models could be used to generate representative (but artificial) traffic. Negative examples representing all other applications are also required; otherwise the number of false positives would be high. It is not possible to construct this part of the training data set as we do not know what other traffic is in the historic trace, and the number of possible applications is simply too large. Therefore, we propose a simple approach that takes the negative examples from the historic trace itself.

We find that individual applications, even those of the same application category (e.g. eDonkey, BitTorrent, Kazaa) can be separated with accuracies of over 90%. Traffic characteristics of the applications have not changed significantly when comparing a hand-classified training dataset with two historic datasets, but the hand classified dataset also shows some limitations, as it does not sufficiently cover the whole range of flow characteristics present in the historic traces. Furthermore, we show that a significant amount of traffic of the applications of interest was using non-default ports and thus would have not been discovered with a purely port number based approach.

The paper is structured as follows. Section II provides a brief overview about related work. Section III outlines our approach, explains our machine learning technique and describes our datasets. Section IV presents the results of evaluating the proposed approach and Section V concludes and outlines future work.

II. Related Work

A number of researchers have proposed machine learning or statistical clustering techniques to separate network application types based on traffic statistics. In [3] the authors use nearest neighbour (NN) and linear discriminate analysis (LDA) to map different applications to different QoS classes. The Expectation Maximization (EM) algorithm is used in [4] to cluster flows into different application types. The authors of [5] have used correlation-based feature selection and a Naïve Bayes classifier to differentiate between different application types. The authors of [8] use principal component analysis (PCA) and density estimation to classify traffic into different applications. We have proposed an approach for identifying different network applications based on forward feature search and EM in [7]. The authors of [6] have developed a method that characterises host behaviour on different levels to classify traffic into different application types.

III. Approach

We use a hand-classified packet trace as a representative sample of the applications of interest. This trace has been separated into classes using payload information and port numbers. Two other public traces without payload serve as historic traces. For the historic traces we obtain the classes based on the application default port numbers, assuming that a major amount of traffic of each application would have used default ports ([2] confirms this for p2p traffic).

Initially we investigate if different applications, especially of the same 'category', can be separated purely based on traffic characteristics by performing 10-fold cross-validation on each trace separately. *k*-fold cross validation randomly divides the data into *k* subsets. Each time, one of the *k* subsets is used as the test set and the other *k*-1 subsets form the training set. Error statistics are calculated across all *k* trials. Cross-validation provides a much better indication of how well the classifier will perform on unseen data than the overly optimistic accuracy obtained on the full training dataset.

We then evaluate how effective a classifier trained on the hand-classified trace can identify traffic flows in the historic traces. We also train classifiers on each of the historic traces and test against the other. Our hand-classified dataset is limited in terms of size and lacks a suitably large range of flow characteristics. Therefore, we also combine the hand-classified data with each historic trace into one dataset and perform 10-fold cross-validation, computing accuracy separately for instances from the hand-classified and historic traces to evaluate if application characteristics have changed.

Our final goal is to estimate the amount of traffic of the applications of interest in the historic trace. As the hand-classified trace does not seem to cover the whole range of application characteristics, in these tests we use the traffic on default application ports of the historic traces as training data.

We also need an additional class of negative examples, into which traffic not of interest will be classified. Without this class, flows from all applications would be classified as one of our applications of interest (whichever is most similar). It is not possible to construct this part of the training data set as we do not know what other traffic is in the historic traces, and the number of possible applications is simply too large.

Dealing with missing negative examples is also known as one-class classification problem and is not unique to our situation [9]. In general, one-class classification is harder than conventional classification because only one side of the class boundary is supported by examples. The two common approaches to this problem are: the generation of artificial negative training instances or the usage of special one-class classifiers [9]. In some preliminary tests we found that none of the approaches worked very well with our dataset.

Therefore we use a different approach, in which negative examples are taken from the historic traces. We first train a classifier on the applications of interest and all other port numbers with a significant number of flows (each port as separate class). We then examine the crossover between the applications of interest and all other ports. The crossover is defined as sum of false negatives and false positives rates between the two classes (see appendix).

If a particular port has maximum crossover with one of the applications we assume traffic flows on this port is of that application and we include the traffic into the application class (positive examples). If a particular port does not crossover with any of our applications we assume that traffic flows on this port is from a different application and include the traffic into the other class (negative examples).

Finally, we test the classifier on the whole historic datasets and compare the amount of application traffic identified using machine learning with the amount of traffic on the default ports.

A. *Flow Attributes*

We use NetMate [10] to compute the flow characteristics (features) based on packet traces. We classify packets to flows based on source IP and source port, destination IP and destination port. Flows are bidirectional and the first packet seen by the classifier determines the forward direction of the flow.

Flows have limited duration. UDP flows are terminated by a flow timeout, while TCP flows are terminated upon proper connection teardown (TCP state machine) or after a timeout (whichever occurs first). We use a 600 second flow timeout as short timeouts such as 60 seconds can cause few long-lived TCP connections to be chopped into multiple flows. In our analysis we consider only UDP and TCP flows that have at least 1 packet in each direction and transport at least 1 byte of payload. This excludes flows without payload (e.g. failed connection attempts) or 'unsuccessful' flows (e.g. requests without responses). Flows devoid of payload data do not reveal information about the generating application. Flows with only a single packet do not resemble a successful communication (without payload inspection it is difficult to determine if these are failed connections or malicious traffic such as port scans).

We compute the following features: protocol, duration, volume in bytes and packets, packet length (minimum, mean, maximum, standard deviation), inter-arrival times (minimum, mean, maximum, standard deviation), active and idle times (minimum, mean, maximum, standard deviation) and the number of pushed packets (TCP only). Aside from protocol and duration all features are computed separately in both directions of a flow. All packet length derived features are based on the IP length excluding link layer overhead. All of our features can be efficiently computed and are packet and flow-level features, meaning packet traces are needed but no wider context (multiple flows) is required to compute the features.

We distinguish active and idle periods by using an idle threshold, which is 1 second by default. Periods where no packets are observed for 1 second or more are treated as idle periods. A flow is active when not in an idle period. When a flow is terminated by timeout the time from the last packet until the timeout is not counted as idle time. Besides computing the volume for the whole flow we also compute the average volume of active periods (bytes and packets).

B. *C4.5 Decision Tree Algorithm*

The C4.5 algorithm [11] creates a classifier based on a tree structure. Nodes in the tree represent features, with branches representing possible values connecting features. A series of nodes and branches is terminated with a leaf representing the class. Determining the class of an instance is simply a matter of tracing the path of feature nodes and branches to the terminating leaf node.

C4.5 uses the divide and conquer method to construct a tree from a set of *S* training instances. If all cases in *S* belong to the same class, the decision tree is a leaf labelled with that class. Otherwise the algorithm will use some test to divide *S* into several non-trivial partitions. Each of the partitions becomes a child node of the current node and the tests to separate *S* are assigned to the branches.

C4.5 uses two types of tests each involving only a single attribute *A*. For discrete attributes the test is A=? with one outcome for each value of *A*. For numeric attributes the test is $A \leq \theta$ where θ is a constant threshold. Possible threshold values are found by sorting the distinct values of *A* that appear in *S* and then identifying a threshold between each pair of adjacent values. For each attribute a test set is generated. To find the optimal partitions of *S* C4.5 relies on greedy search and in each step selects the test set that maximizes the splitting criterion. C4.5 uses the entropy based gain ratio (see [11]) to select the best split.

The divide and conquer approach partitions the data until every leaf contains instances from only one class or a further partition is not possible e.g. because two instances have the same features but different class. If there are no conflicting cases the tree will correctly classify all training instances. However, this over-fitting leads to a decrease of the prediction accuracy on a set of unseen instances.

C4.5 attempts to avoid over-fitting by removing some structure from the tree after it has been built. This pruning is based on estimated true error rates. After building a classifier the ratio of misclassified instances and total instances can be viewed as the real error. But this error is minimised because the classifier was constructed specifically for the training instances. Instead of using the real error the C4.5 pruning algorithm uses a more conservative estimate, which is the upper limit of a confidence interval constructed around the real error probability. With a given confidence CF the real error will be below the upper limit with 1-CF. C4.5 uses subtree replacement or subtree raising to prune the tree as long as the estimated error can be decreased.

We compared C4.5 with other algorithms such as Naïve Bayes and Neural Networks and found it to be the most effective. It provides high accuracy, training and classification are fast and memory usage is moderate. Because C4.5 selects the tests in order of decreasing gain, it already performs feature selection during the training. Important features will appear at the top of the tree and irrelevant features are not used. Therefore C4.5 does not suffer (as e.g. Naïve Bayes) if irrelevant features are present in the data set. Furthermore, the tree output can be easily interpreted by humans (although this is complicated for large trees) or translated into rules, which is another advantage over other algorithms (e.g. neural networks are essentially black boxes).

C. *Traffic Traces*

We captured several hours of p2p and other application traffic on an ADSL link in September 2005. The different p2p applications were run independently and the first 68 bytes of each packet were captured to allow for later 'hand classification' of the flows. We used the following p2p applications: eDonkey (eMule 0.46c), Kazaa 3.0 and BitTorrent 4.0.4. All p2p applications were run on the preconfigured default ports.

Table 1: P2P application signatures

P2P Application	Signatures	Protocol	Default ports
eDonkey (EMule 0.46c)	0xe3, 0xe4, 0xe5, 0xc5, 0xd4 (first byte)	TCP UDP	4661-4666 4671-4673
BitTorrent 4.0.4	0x13BitTorrent, GET /announce?info_hash	TCP UDP	6881-6889
Kazaa 3.0	GET /.hash, GIVE, GET /.sig, POST /.pkt, GET /.file	TCP UDP	1214, 3415 3531

Table 2: Traffic statistics of applications of interest in traces used for training

Application	**Payload Classified (PC)**		**Twente**		**Leipzig**	
	Flows	**MBytes**	**Flows**	**MBytes**	**Flows**	**MBytes**
Edonkey	3062	2,325.7	50,000	10,032.8	50,000	32,833.1
BitTorrent	3216	856.7	32,261	4,121.7	NA	NA
Kazaa	882	524.7	47,893	3,308.0	50,000	7,360.6
Half-Life	1902	592.8	50,000	211.8	50,000	3,688.3
HTTP	7894	221.2	50,000	993.1	50,000	1,461.8
HTTPS	560	67.9	27,835	247.2	50,000	372.6
DNS	143	2.1	21,115	14.3	50,000	60.4
Total	17,659	4,591.1	229,104	18,928.9	300,000	45,776.8

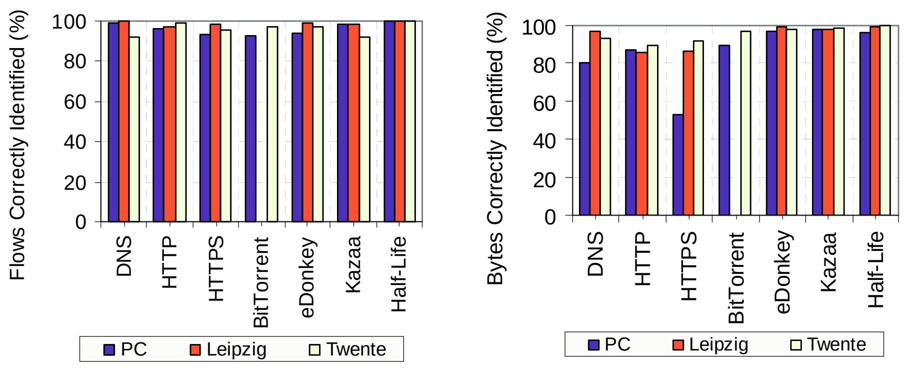


Figure 1: Percentage of flows (left) and bytes (right) correctly classified performing 10-fold cross-validation separately for each trace. Note: Leipzig trace does not contain BitTorrent.

As in [2] we use a mixture of port numbers and signatures to identify the p2p traffic. Interestingly the application signatures as identified in [2] were not able to detect all the p2p traffic. The old signatures were still valid (as far as they appeared in our trace) but we found a number of new signatures, most likely a result of new client and/or protocol versions (see Table 1). This shows that when using signatures care must be taken that the signatures are accurate and complete.

We also captured traffic of the game Half-Life 1, as a representative for online games, as well as HTTP, HTTPS and DNS traffic, as these are among the most prominent network applications.

We use two publicly available traces as our historic data. We use the first 8 days of [12] (location 4) captured in February 2004 and one day of Leipzig-II [13] (leipzig-ii-20030221) captured in February 2003. Both traces are anonymised and packet payload information had been removed. As these traces contain many flows they were randomly sampled taking up to 50,000 flows for each application based on the default ports for the training datasets. The Leipzig trace does not contain any BitTorrent traffic.

The eDonkey protocol generates a large amount of flows with very small volumes and only few flows with larger volumes, which causes several problems. Firstly, eDonkey produced substantially more flows than any other application in the hand-classified dataset. Secondly, sampling of the Leipzig and Twente traces for eDonkey obtained almost only small flows. Therefore, for all traces we have generated the eDonkey classes by sampling 50% large flows and 50% small flows.

Table **2** shows the data used for training the classifier to the applications of interest. We also randomly sampled 350,000 other flows from each historic trace of which we select the negative training examples.

IV. Evaluation

Classification and training is performed using the Weka Machine Learning suite [14], and its implementation of the C4.5 algorithm, J48. We use a confidence level of 0.25 and the minimum number of instances per leaf is 2. We use subtree replacement and subtree raising when pruning.

A. *Separating the Different Applications*

We initially determine if each of our nominated applications can be separated from each other. We perform training and 10-fold cross validation on each dataset separately. Figure 1 shows the percentage of flows and bytes that was correctly identified. The results show high accuracy for all applications and datasets. Overall flow accuracy was very high, with 99.75% of the PC trace flows being identified, 96.19% of Twente flows and 98.84% of Leipzig flows.

For the three traces, all applications were identified with greater than 90% accuracy. The results for byte accuracy are less consistent, but quite good for the p2p applications. HTTPS has very low byte accuracy in the PC trace because most of the volume was in a single large flow that was misclassified.

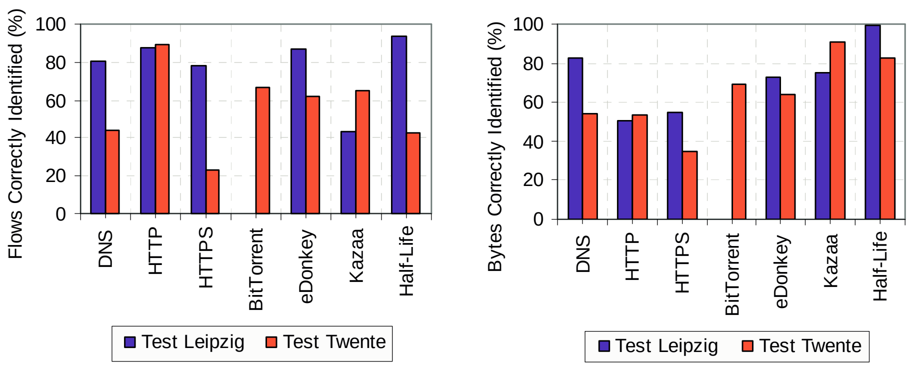


Figure 2: Percentage of flows (left) and bytes (right) correctly classified when trained on PC and tested on Twente or Leipzig. Note: Leipzig trace does not contain BitTorrent.

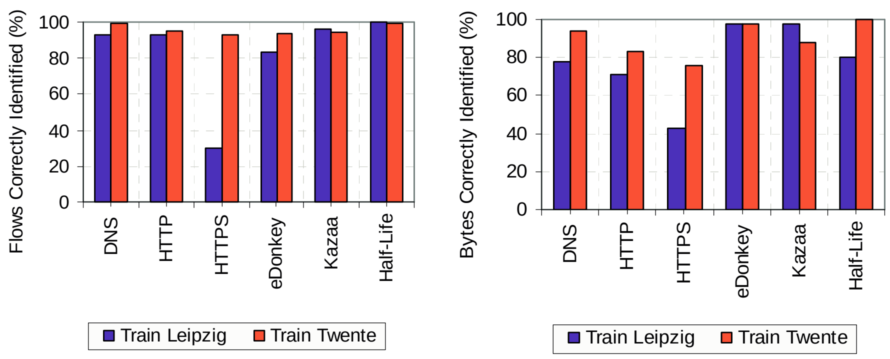


Figure 3: Percentage of flows (left) and bytes (right) correctly classified when training on Twente or Leipzig and testing on the other.

B. *Evaluating the Similarity of Application Characteristics Across Traces*

To investigate if the traffic characteristics of the different applications are similar between the recent and past traces we train on the PC dataset and test the classifier against the Leipzig and Twente datasets. Figure 2 shows the flow and byte accuracies.

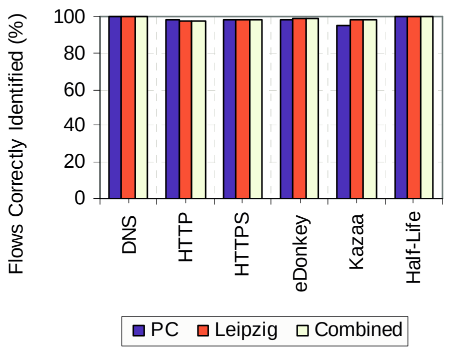
When training on the payload classified dataset and testing on Twente flow-based accuracy is not very high. However, this may be due to a lack of training instances for some of the applications (such as DNS), as accuracy of the different applications appears somewhat proportional to the number of flows in the PC dataset. Byte accuracy improves somewhat for the p2p and Half-Life traffic classes. While only 40% of Half-Life flows were correctly identified, these represented more than 80% of the volume. This indicates that the majority of correctly classified instances were actual game flows but a significant number of small probe flows were misclassified (see [15] for a discussion on game vs. probe flows). Accuracies for Leipzig were generally better than for Twente, except for Kazaa.

It seems that our hand-classified dataset is limited in terms of the number and representative quality of instances and therefore unable to predict the data in the historic traces with very high accuracy. The results do show however a significant correlation between classes in the training and testing sets, and a larger training set would likely produce better results.

To investigate if the traffic characteristics of the different applications are similar between the historic traces we trained on each historic dataset and tested against the other. Figure 3 shows the flow and byte accuracies. The increased size and variance within the training classes provides some increase in classification accuracy, when compared to training on the PC dataset. In general training on Twente and testing on Leipzig provides better results, though both scenarios identified p2p and game traffic quite accurately with 80-90% accuracy.

The lower prediction accuracy of the PC training data does not necessarily mean that application characteristics have significantly changed. To test this we combine the PC dataset with each of the historic traces and perform 10-fold cross validation on the combined sets. We compute not only the overall accuracy but also the accuracy for PC and historic dataset separately.

Figure 4 shows the results when combining the PC and Leipzig datasets. Overall accuracy and the accuracies for the individual datasets are obtained by separating the instances after cross-validation. The results obtained for PC and Leipzig when separated are essentially the same as in Section IV.A, with the PC dataset actually seeing increased accuracy for HTTP/HTTPS. DNS also increases markedly for PC, clearly benefiting from the additional DNS instances in the Leipzig trace.



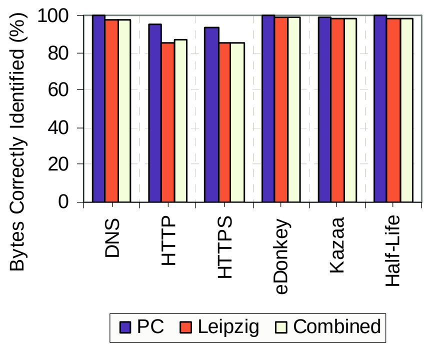
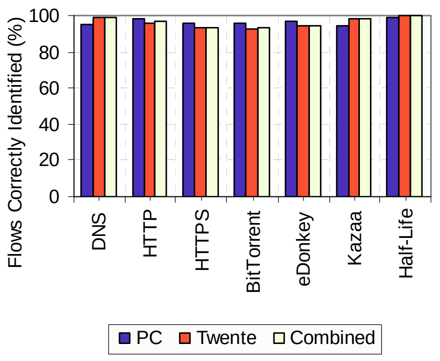


Figure 4: Overall and separate flow (left) and byte (right) accuracy statistics for PC and Leipzig traces after combined training and cross-validation.



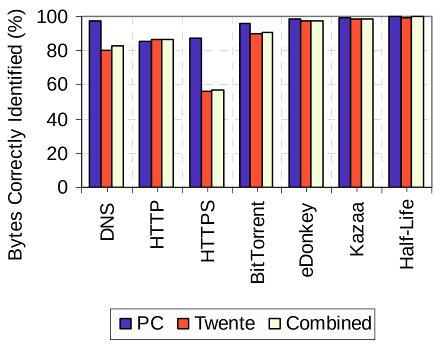


Figure 5: Overall and separate flow (left) and byte (right) accuracy statistics for PC and Twente traces after combined training and cross-validation.

Figure 5 graphs the results when combining PC and Twente datasets. Again there is little difference in accuracy when compared to individual cross-validation of both datasets, an indication that combining the two training sets did not introduce additional errors, and that the characteristics of all applications are very similar between the datasets.

To test whether the classifier had been over-fitting the datasets (and providing inflated accuracies), we switched the labels of two classes in one part of the combined set (i.e. either the PC or historic) and re-ran the cross-validation tests. As expected changing the labels greatly reduces accuracy. If classes differ in size between traces the majority class only experiences a small accuracy reduction, whereas the minority class suffers from a huge accuracy decrease.

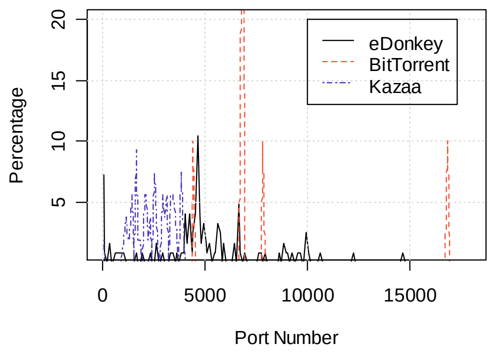
C. *Estimating Traffic of Interest in the Historic Traces*

In this section we focus on p2p applications and Half-Life, as in our set of applications these are the most likely to run on non-default port numbers. We obtain positive training examples solely from the default ports in the historic traces. To estimate the presence of p2p and Half-Life traffic within the anonymised traces we must also generate an 'other' class containing negative examples.

We do this by calculating the crossover between the applications classes and other classes representing individual ports within the historic traces. As there are many thousand ports used in the traces, we remove ports with less than 25 flows from each dataset (due to limited computing resources). Using our default-port application classes and each other port as a class, we train a classifier and perform cross-validation testing. If the maximum crossover between two classes is higher than 0.5 we integrate both. (A value of 0.5 is equivalent to 25% false negatives and false positives between the classes, which is a much higher error than found in Section IV.A and IV.B.) After integrating highly similar traffic on non-default ports into our p2p and Half-Life classes a total of 249,586 and 303,307 instances were still left in the 'other' class for the Leipzig and Twente traces respectively (and were used as negative examples).

Figure 6 shows histograms of ports that were identified as being potential members of the p2p classes for the Twente and Leipzig traces (few ports >20000 excluded). A bin size of 100 is used in both histograms. There is a clear distinction between the identified ports for each application, and the distributions are fairly similar for both traces. Similar distributions were also found for the payload-classified trace (not shown).

Kazaa appears to have a large number of flows on non-default ports mostly distributed across ports 1000-4000 (default port 1214). eDonkey is found on a wider distribution of ports, though most are concentrated on either side of the default ports (around 4662). In the Twente trace BitTorrent is centered mostly on the default port (6881), with 40% of flows occurring in this region (peak not shown in figure).



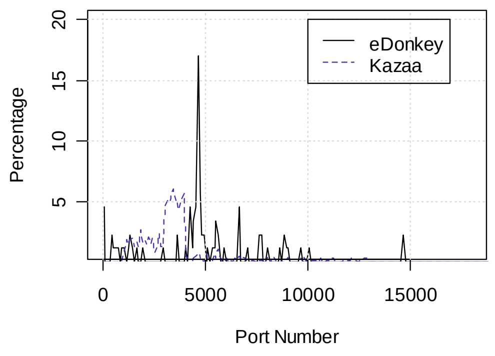


Figure 6: Histogram of non-default peer-to-peer port numbers for Twente (left) and Leipzig (right).

Figure 7 plots the distribution of non-default Half-Life ports for the historic datasets. Again a bin size of 100 is used. We see that the majority of game flows on non-default ports are in the space immediately above the default port (27015). Interestingly, few servers were apparently running on 'multiples' of the default port, such as 28015 or 29015.

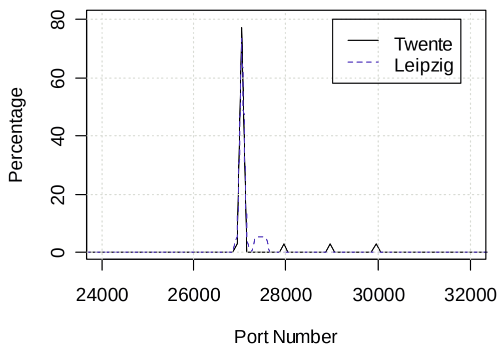


Figure 7: Histograms of Half-Life non-default port distributions for the Twente and Leipzig traces.

After aggregating the identified ports into either the application classes or the negative examples class, a classifier was built and tested on each respective historic trace (detailed in Table 3).

Table 3: Total number of flows and volume of two historic traces.

Trace	Flows [M]	Volume [GB]
Twente	3.4M	79.3GB
Leipzig	9.1M	238.8GB

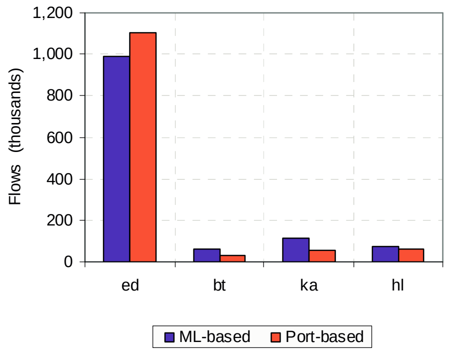
Figure 8 compares the application traffic estimates for the Twente trace using default port analysis and ML-based analysis. Port-based analysis underestimates the number of flows for each of the applications other than eDonkey. In terms of volume, ML-based analysis uncovers 13GB of potential p2p traffic that remained unidentified using port-base analysis. This represents approximately 16% of the trace volume. Kazaa appears to have the largest amount of traffic on non-default ports, as was suggested by the wide port distributions. 37.9GB (47%) of traffic was predicted as other applications.

Figure 9 compares the application traffic estimates for the Leipzig trace using port-based analysis and ML-based analysis. Again port-based analysis underestimates the number of applications flows, except eDonkey. As with Twente, ML-based analysis shows significantly more volume than was identified using port-based analysis, with 36GB of additional p2p traffic, or about 15% of the trace volume. 72GB (30%) of the traffic was predicted as other applications.

Table 4 shows the traffic volume of each application on the default ports, and the percentage this volume represents out of the total volume estimated by the ML classifier (as given in Figures 8 and 9). It shows that port-based analysis would have significantly understated application flows and volume. Even for Half-Life 20-30% of the traffic would have not been detected. There are some similarities in the results obtained, for example eDonkey and Half-Life have the highest and Kazaa has the lowest percentage of traffic on the default ports for both historic traces. We also find the same trends for the hand-classified trace, although the percentages are different as port usage is biased towards a specific client (e.g. all p2p applications were run on default port numbers).

Standing out against the overall trend is the predicted number of eDonkey flows when using ML-based classification. For both Twente and Leipzig, the ML classifier predicted 10% less flows than found using port-based filtering. Further analysis found the cause to be a combination of the unique behaviour of the eDonkey protocol and a weakness in the current methodology.

As discussed in Section III.C, the eDonkey protocol generates significantly more flows than the other applications, the majority of which are small-volume signalling flows. By restricting the sample size to 50,000 flows, the eDonkey training dataset is a much smaller fraction of the total flow population compared to the other applications. Different eDonkey/eMule signalling protocol operations [16] occur at different frequencies and it is likely that the less frequent operations would be underrepresented after sampling. As our training examples were sampled with a 50/50 split of large and small volume flows, the variety of training examples for the signalling flows is further reduced.



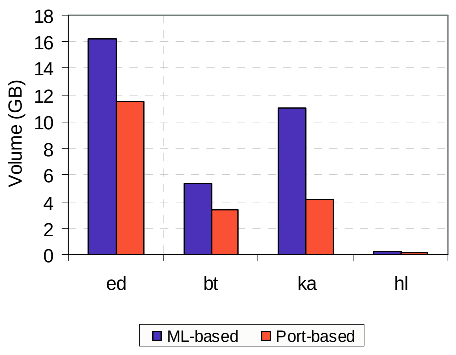
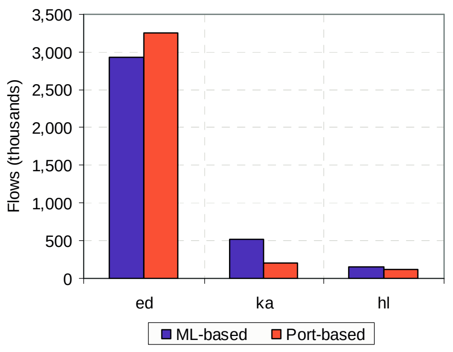


Figure 8: Traffic estimates in terms of flows (left) and volume (right) for Twente trace using port-based and ML-based methods.



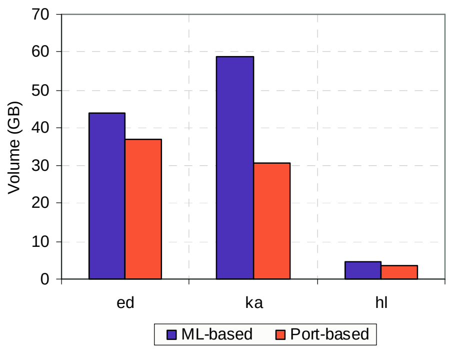


Figure 9: Traffic estimates in terms of flows (left) and volume (right) for Leipzig trace using port-based and ML-based methods.

Table 4: Traffic on default ports (as percentage of ML estimate)

	Traffic on default ports (Twente estimate)		**Traffic on default ports (Leipzig estimate)**	
	Flows	**Volume**	**Flows**	**Volume**
EDonkey	1.1M (111%)	11.5GB (70.9%)	3.3M (110%)	36.9GB (83.9%)
BitTorrent	33K (55.6%)	3.36GB (63.1%)	-	-
Kazaa	51.4K (44.6%)	4.1GB (37.8%)	207.1K (39.5%)	30.7GB (52.3%)
Half-Life	60.7K (86.9%)	216.7MB (71%)	124.2K (80.3%)	3.7GB (82.8%)

Our method of selecting the 'other' class only selects training instances on a port-by-port basis. We found that some ports contain eDonkey signalling among a much larger number of flows from an unknown application (according to the port number it could be Gnutella). The PC trace confirms that eDonkey traffic occurs on these ports. Although the eDonkey flows only represented a small portion of the total flows on these ports their number is slightly larger than the number of similar flows in the eDonkey training class. Therefore, many of these signalling flows in the test data – including those on the default port – are classified into the 'other' class (although with low prediction probability around 60%).

D. *Limitations*

An obvious drawback with our approach of obtaining negative training examples is that it only selects on a port-by-port basis and thus cannot properly deal with ports that are significantly used by multiple applications of interest and/or the 'other' class. This problem occurred for the eDonkey class, which has a significant number of flows on ports whose predominant application is unknown (see previous section).

Another drawback is that due to processing and memory limits we cannot perform crossover analysis on all ports within a dataset but must focus only on the most prominent ones.

V. Conclusions and Future Work

In this paper we have shown that individual applications can be separated with high accuracy without using port numbers or payload information even if they are of similar type (e.g. different p2p file-sharing applications). We have shown that the traffic characteristics of the applications have not significantly changed in the time between capture dates of the traces used. We have also proposed a method for obtaining the negative examples required for training a classifier. Our method takes into account the traffic mix in the historic traces and the fact that the traffic of interest is spread across many ports other than the default.

The ML-based classifier detected a significant amount of traffic that would have not been detected purely based on port numbers. The non-default port number distributions found for the different p2p and game applications provide some circumstantial evidence of the efficiency of our method, as there is clearly some correlation between non-default port numbers and the actual application.

A number of issues need to be addressed in future work. Currently our evaluation is limited by the fact that we lack access to historic traces with packet payloads necessary to validate our results. We have not yet investigated the effects of different parameters such as different crossover threshold values etc. and our method for identifying non-default ports has some limitations that need to be improved in the future. We also plan to investigate better ways of obtaining training data for applications with a wide variance of different traffic flows, such as eDonkey.

ACKNOWLEDGMENTS

This paper has been made possible in part by a grant from the Cisco University Research Program Fund at Community Foundation Silicon Valley.

REFERENCES

1. T. Karagiannis, A. Broido, N. Brownlee, k claffy, M. Faloutsos, “File-sharing in the Internet: A characterization of P2P traffic in the backbone”, Technical report, November 2003.
2. Thomas Karagiannis, Andre Broido, Nevil Brownlee, kc claffy, “Is P2P dying or just hiding?”, Proceedings of Globecom 2004, November/December 2004.
3. M. Roughan, S. Sen, O. Spatscheck, N. Duffield, “Class-of-Service Mapping for QoS: A statistical signature-based approach to IP traffic classification, ACM SIGCOMM Internet Measurement Workshop 2004, Taormina, Sicily, Italy, 2004.
4. A. McGregor, M. Hall, P. Lorier, J. Brunskill, “Flow Clustering Using Machine Learning Techniques”, Passive & Active Measurement Workshop 2004 (PAM 2004), France, April 19-20, 2004.
5. Andrew W. Moore and Denis Zuev, "Internet Traffic Classification Using Bayesian Analysis Techniques", ACM SIGMETRICS June 2005, Banff, Canada.
6. T. Karagiannis, K. Papagiannaki, and M. Faloutsos, “BLINC: Multilevel Traffic Classification in the Dark”, ACM Sigcomm, Philadelphia, PA, August 2005.
7. S. Zander, T.T.T. Nguyen, G. Armitage, "Automated Traffic Classification and Application Identification using Machine Learning", IEEE LCN 2005, Sydney, Australia, 15-17 November 2005.
8. T. Dunnigan, G. Ostrouchov, “Flow Characterization for Intrusion Detection”, Oak Ridge National Laboratory, Technical Report, http://www.csm.ornl.gov/~ost/id/tm.ps, November 2000.
9. D. M. J. Tax, R. P. W. Duin, “Uniform Object Generation for Optimizing One-class Classifiers”, Journal of Machine Learning Research 2, pp. 155-173, 2001.
10. NetMate, http://sourceforge.net/projects/netmate-meter/ (as of March 2006).
11. R. Kohavi and J. R. Quinlan, “Decision-tree discovery”, In Will Klosgen and Jan M. Zytkow, editors, Handbook of Data Mining and Knowledge Discovery, chapter 16.1.3, pages 267-276, Oxford University Press, 2002.
12. Remco van de Meent, M2C Measurement Data Repository, University of Twente, Enschede, The Netherlands, http://m2c-a.cs.utwente.nl/repository/, December 2003.
13. NLANR traces: http://pma.nlanr.net/Special/ (as of March 2006).
14. WEKA 3.4.4, http://www.cs.waikato.ac.nz/ml/weka/ (as of March 2006).
15. S. Zander, D. Kennedy, G. Armitage, ”Dissecting Server-Discovery Traffic Patterns Generated By Multiplayer First Person Shooter Games”, ACM NetGames 2005, NY, USA, 10-11 October 2005.
16. Y. Kulbak, D. Bickson “The eMule Protocol Specification”, Technical Report, Distributed Algorithms Networking and Secure Systems Lab, The Hebrew University of Jerusalem, Jerusalem, January 2005.

APPENDIX

A confusion table provides the basis for the evaluation metrics and contains information about actual and predicted classifications for each class. Table 5 shows the confusion matrix for a two-class classifier (classes - and +).

Table 5: Confusion matrix

Label	Assignment	
	-	+
-	TN	FP
+	FN	TP

Each instance is associated with a label, which accounts for the true class. The classification produces an assignment indicating whether it believes an object to belong to a certain class.

Then for each instance there are four possible outcomes: TP stands for true positive, TN stands for true negative, FP stands for false positive and FN stands for false negative. The false positive rate and false negative rate are defined as:

$$FP\_rate = \frac{FP}{TN + FP}, \quad FN\_rate = \frac{FN}{FN + TP}. \tag{1}$$

The maximum crossover of a class x with all other classes from a set of classes C is:

$$crossover(x) = \max\{\forall c \in C, c \neq x \mid FP\_rate(x,c) + FN\_rate(x,c)\}. \tag{2}$$