Efficient p2p traffic classification in a campus environment

Reinette Chatre, Jim Binkley, Suresh Singh, Tim Menzies, Andres Orrego

Abstract

Proper control of network traffic requires understanding which applications are using the network. In particular, identifying p2p traffic is of particular importance since it represents a rapidly growing fraction of all traffic. Prior work on classification of peer-to-peer (p2p) traffic relied heavily on pattern matching on application (layer-7) data. Such classifications are time consuming to generate and will become outdated whenever new applications appear or old applications use more encryption. Our method uses statistical attributes of traffic to identify p2p sources and as such is immune to encryption. In this paper, we first we identify several easy to compute layer-3 and layer-4 attributes that enable accurate application classification. Second, our algorithm builds a classifier based on an oracle and a one-pass Bayes algorithm that supports rapid classification of data. This means that the classifier is built automatically and does not require any manual input or training. The classifier runs in real-time and we show that, on certain key measures, our algorithm outperforms prior results. We achieve high probabilities of detection with a remarkably low false positive rate resulting in an accuracy of classification of between 70% and 98% in most cases.

Key words: Data mining, machine learning, traffic classification, p2p, Naive Bayes classifiers, discretization.

[1] Reinette Chatre, Jim Binkley, Suresh Singh, Tim Menzies, are with the Computer Science Department, Portland State University. Email: rchatre@cs.pdx.edu,tim@timmenzies.net, jrb@cs.pdx.edu, singh@cs.pdx.edu

[2] Andres Orrego is with Global Science and Technology, Inc. Email: Andres.S.Orrego@ivv.nasa.gov

[3] Suresh Singh is the **corresponding author**, singh@cs.pdx.edu, Tel: 001-503-725-5402, Fax: 001-503-725-4211, 1900 SW 4th Avenue, Suite 120, Portland State University, Department of Computer Science, Portland, OR 97207, U.S.A.

1 Introduction

Reliable traffic classification is important for every network. Ensuring that applications with legitimate bandwidth needs and requirements are accommodated is important for any business relying on these applications for day-to-day and core business operations. On a grander scale the growth and dynamics of bandwidth usage is critical for network operators that need to deliver on service level agreements and prepare for future growth.

Knowing how bandwidth is used by applications relies on accurate identification of these applications. In this paper we explore the accurate identification of applications by focusing on p2p applications. The speed of growth (in usage numbers and bandwidth consumption) and the versatility of the developer and user community makes these (p2p) applications particularly interesting. Analysis [12] on network traffic from a tier-1 ISP at a time (2001) when port numbers were reliable for traffic classification showed that a significant portion of network bandwidth is consumed by p2p applications. Each host could generate anywhere from 1.6MB/day to 19.6MB/day. The growth of the p2p bandwidth was particularly interesting: in four months the combined bandwidth used by p2p applications more than *doubled* - from 1032GB/day to 2089GB/day [12]. The general consensus based on empirical evidence is that this rate of growth is continuing. p2p applications clearly contribute a significant amount of traffic and any site performing bandwidth management needs to pay close attention to this category of application.

Unfortunately, empirical data and recent investigations [8] show us that the traditional usage of TCP and UDP port numbers to identify applications on the network has become unreliable. Port numbers were reliable for p2p detection as recently as 2001, since then their usage for p2p classification dropped significantly. Analysis on network traffic from 2003 [11] reports that the usage of port numbers could only classify 30% (in the worst case) of p2p traffic. Signature based classification schemes based on pattern matching are still very accurate but, given the trend towards encrypted payloads [3], these methods will also be unable to classify p2p traffic in the future. Thus, several researchers, including ourselves, have started exploring machine learning techniques as a way to classify p2p traffic.

1.1 Research Contributions

Our solution for p2p classification works at the edge of a network and is able to classify which internal hosts participate in p2p applications as this (the participation) occurs. Our approach is to use fully automated machine

learning to build a p2p classifier. Indeed, our results show that this approach achieves an accuracy of classification better than other existing methods and does so in real-time. More importantly, our approach has the following features which are absent in prior work:

Host classification: We classify which *hosts* are participating in p2p application. We do not perform individual flow classification because we believe that it is as informative (sometime more useful to network managers) to know if a host participates in p2p applications than to know which flow of a particular host is occupied by a p2p application.

Reproducibility: A difficulty with commenting on prior results is that the data used in their analysis is proprietary. We take care to base our work on public domain data sets [2] so that other researchers can repeat or refute or improve our results. We also create our training data using open source tools that can be used by anyone needing to recreate our work, these tools are described in Section 3.

Automation: Prior results required an elaborate manual pre-classification of the input examples. Our system works 100% automatically.

Specialization: We refine prior comments on the merits of specialized descriptors. Moore and Zuev [9] described experiments where important subsets of the data were identified. Here, we reproduce and extend those prior results.

The remainder of this paper is organized as follows. Section 2 initiates our discussion with a description of our research criteria. Section 3 describe the components of our solution. Section 4 presents a survey of current research in traffic classification. Section 5 describes our selected attributes. Section 6 describes a series of experiments that explore the behavior and accuracy of our classifier. Section 7 then describes the real-time version of this classifier and its performance. Section 8 discusses how our approach can be extended to the case when all p2p traffic is encrypted. Section 9 ends this paper with a discussion of the results and future work.

2 Measures for the Quality of Classification

In order to quantify the performance of our algorithm appropriately, we complete a *confusion matrix* during every test. An example is given in Table 1. This table lets us define the *accuracy* of the classifier as the number of correct classifications seen over all classifications: $accuracy = \frac{a+d}{a+b+c+d}$. We require two valuable measures to evaluate our performance: *probability of detection* (PD) and *probability of false positive* (PF). If the classifier detects a p2p application

Classifier prediction	Real ("oracle") Classification	
	nominal	p2p
nominal(not p2p)	a	b
p2p	c	d

Table 1
Confusion matrix

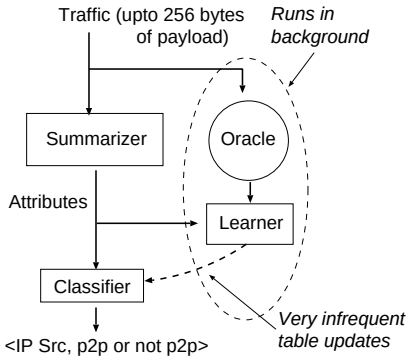

Fig. 1. Overall approach

there are two cases of interest. In one case the classifier correctly identified a p2p instance. This PD is the ratio of all correctly detected p2p instances to all actual instances of p2p applications: PD= $\frac{d}{b+d}$. In the second case when the classifier detects a p2p application when none is present corresponds to false positive. Thus we define PF as the ratio of the number of p2p applications identified when there are none present: PF= $\frac{c}{a+c}$. Ideally our solution will have a high PD and very low PF.

Note that systems can have high accuracies but low probabilities of detection. For example, consider a system where 10% of its traffic is p2p and the classifier never triggers (perhaps due to some internal fault). This classifier would score an accuracy of 90% even though its PD was 0%. Hence, it is necessary to use PD in addition to accuracy as a measure of quality.

3 Research Approach

Figure 1 illustrates our approach which consists of two parts – a training component and the classifier. Traffic captured by the frontend is fed to the *summarizer* in 30 second chunks. The summarizer outputs a tuple $<$*IP src, attr_1, attr_2,* $\cdots >$ which consists of attribute values for each IP source seen. The attributes we use are discussed in detail in section 5 but examples may include number of ICMP errors returned to that IP src, number of SYNs seen, etc. We note that while the traffic capture includes 256 bytes of payload per

packet, this payload is only used by the *oracle* and completely ignored by the *summarizer* and *classifier*. The attribute tuples produced by the summarizer are fed to the *classifier* which in turn classifies each IP source as either running a p2p application or not. The IP source address is not used by the classifier to make a decision, just the attributes associated with it. The classifier runs very fast and uses a simple table-based algorithm discussed later. The table itself is produced by the *learner* in the background.

Periodically, traffic captured by the summarizer is also fed to the *oracle* which uses the payload to classify the traffic as p2p or not. This information, along with the attribute tuple produced by the summarizer, are fed to the learner which runs a Naive Bayes learning algorithm and outputs the table used by the classifier. The key point to note here is the following: *the classifier only receives updates from the learner very infrequently (every time a new p2p application shows up or when we discover that the accuracy of the classifier has degraded) thus the speed of operation of the classifier is not hindered in any way by the learner*. Indeed, our clasisfier runs in real-time.

The *summarizer* is implemented on FreeBSD. The oracle is based on a combination of the *l7-filter* [1] and pattern matching scripts developed in-house. The l7-filter is a classifier for Linux's Netfilter that identifies applications using application data regardless of port. Our pattern matching scripts enhance the ability of the l7-filter by using header information in addition to string matching of the application data to identify specific p2p applications. In combination, these two tools are almost completely accurate in detecting p2p applications in addition to other applications such as servers (email, web), botnets, scanners and irc (see also section 5 where we have examples of how the selected attribute values depend on the underlying applications). However, as a practical matter these tools are very slow to run and thus cannot be used in anything approaching real-time (and obviously cannot work if the payload is encrypted).

The *learner* consists of two components: an incremental discretizer (which puts values for a given attribute into appropriate buckets of the appropriate size) and a Naive Bayes learner. The discretizer uses a novel one-pass method ideal for large data sets as it scans the input only once and automatically adjusts the bin sizes to accomodate and capture changes in the data being discretized. It also makes no apriori assumptions of the underlying numeric values being discretized. In our case, each training data set consists of 10 million packets so such a discretizer is very useful.

Naive Bayes classifiers are based on Bayes' Theorem. Informally, the theorem says $next = old * new$ i.e. what we'll believe *next* comes from how *new* evidence

effects *old* beliefs. More formally:

$$P(H \mid E) = \frac{P(H)}{P(E)} \prod_i P(E_i \mid H)$$

i.e. given fragments of evidence E_i and a prior probability for a class $P(H)$, the theorem lets us calculate a posteriori probability $P(H \mid E)$. Our learner runs the discretizer on a data set and then runs the Bayes classifier to create a likelihood table for each attribute. The table simply consists of two columns labeled 'p2p' and the other 'nominal' (i.e., not p2p). Each row corresponds to a bin created by the discretizer and the numeric values in each table entry (say $(i, \text{p2p})$) is a frequency count of the number of IP sources for which the given attribute took a value in bin i and was classified by the oracle as being p2p. This set of tables are then fed to the classifier. When the classifier is given attribute values for a IP source, it simply uses Bayes theorem above to compute the likelihood of the IP source being p2p or nominal. The tables provide the conditional probabilities needed in the Bayes formula. It is therefore easy to see why the classifier works very fast.

We note that the above approach requires p2p traffic to be unencrypted so that the oracle can properly classify it (to train the classifier). In section 8 we discuss a simple extension of the above approach to deal with the case when all p2p traffic is encrypted.

4 Related work

The presence of p2p traffic in the Internet has been drawing more and more attention from researchers and network operators in recent years due to the impact this category of application can have on a network resulting from significant growth and unreliable profiling. The work in [12] investigated the behavior of p2p traffic in an attempt to understand its impact on a large ISP network. The authors note in their work that the data on which their research was based was captured while the p2p applications could still be found through the usage of well known port numbers. Conducting the same work in a network today will not produce accurate results as the p2p applications will not be utilizing well known ports.

With the loss of port numbers researchers moved towards traffic classification through application signatures inside the payload. The work described in [11] created signatures for five different p2p applications and implemented a system dedicated to p2p classification. The work resulted in false negative ratio of less than 5% for most protocols (about 10% for bittorrent). A valuable result is their comparison with port-based classification where significant portions, up

to 72.6% in one case, of p2p traffic was detected on non-standard ports. Traffic classification using application signatures [11] will only work while we, (1) have reliable signatures for most p2p applications, and (2) the p2p applications do not utilize any encryption. Both cases are expected to disappear in the near future and thus *traffic classification needs to be performed without relying on the port numbers and without access to the payloads transmitted between hosts.*

The authors of [6] manually created unique heuristics for p2p detection using domain knowledge of transport protocol characteristics and connection patterns. Enhancements to these heuristics can be found in [7] where the hosts, rather than their flows, are explored for their behavior on *social, functional,* and *application* levels in order to classify hosts' participation in one of several traffic categories - one of which is p2p. Our solution determined the attributes relevant to p2p application identification through automation. Although domain knowledge was initially applied to create these attributes the decision about their relevance to application detection was determined automatically - no knowledge of p2p behavior was required. We support the authors' conclusion in [7] that it is more intuitive to perform host based classification - where an application is associated with a host - rather than flow based classification. In our solution the path from the actual network traffic through the summarizer to the classifier is clear. With this architecture our traffic classification can be performed *real-time* at the network border. This path is not presented in [7] as the classifications are performed on batches of flow data collected from several devices in the network. Motivated by the *real-time* nature of our setup we maintain minimum state. Our solution does not keep a history of previous classifications in order to direct current classifications. A characteristic related to the goal of minimum state is that our solution very early "forgets" flow data by summarizing this information to form a general picture of *host behavior*. The classifier never sees individual flow data and thus performs the classification on significantly less data resulting in potential speed increases for real-time classification. The authors of [7] reported that the system could classify 98% of the p2p instances with an accuracy of 82%. Values for *probability of detection* and *probability of false positive* data were not reported. The values quoted are for the case where hosts have to participate in at least one flow. This is the case of interest in our work as we would like to detect instances of p2p participation as it occurs on the network. Our solution was able to achieve an accuracy of 93% in the classification of local IP sources. A comparison of execution speeds between the solution in [7] and ours have not been performed as the former is not publicly available.

In parallel to heuristic based traffic classification the research community also explores machine-learning approaches. A Naive Bayes estimator is used in [9] to classify traffic into ten different categories - one of which is p2p. The authors report results from two testing datasets and show trust values (*probability of detection*) of 36.45% and 55.18% respectively. This work addresses *flow*

classification and is different from our technique of *host classification*. We summarize all flow data into a general picture of host behavior and perform traffic classification by examining this host data. While our work is also in the machine learning domain we utilize current research in order to create a solution that can perform real-time classification on live data. The classifier in [9] requires the entire dataset to be in memory implying that the classifier can never be used on real-time data or on very large data sets. A noteworthy result from the work in [9] is the performance of the system after attribute selection has been completed. Using the most relevant attributes the system was able to achieve a *probability of detection* of 36.45% on one testing dataset and 55.18% on another. These variations is a motivation for more attention and will be explored in Section 6.3. Our solution was able to achieve an accuracy of 93% (*probability of detection* of 79.5%) in the classification of local IP sources using datasets that were collected nine months apart.

A technology that is very promising (and also related to an aspect of the work in [7]) is the usage of *communities of interest* to identify the interacting hosts of a p2p network [4]. *Statistical clustering* ([5]), where connections are grouped into traffic classes that represent similar communication patterns, and *statistical signature based classification* using the nearest neighbor and linear discriminant analysis techniques [10] is also addressing the topic of traffic classification without application data.

5 Attributes Used

In order to distinguish p2p traffic from all the other traffic in the network, we need attributes that can, individually or in combinations, discriminate between different traffic types. Several different attributes are used in this study and the selection of these attributes is driven by empirical observation of their effectiveness as well as an intuitive understanding of how the attributes may distinguish between different traffic types. We divide the attributes into two categories – basic network control attributes and attributes that are composed of basic network control data. These attributes are measured from the point of view of an individual IP host over some period of time.

5.1 Basic network control attributes

(1) SYN: number of TCP SYN segments sent. A large number of SYNs from an IP source typically indicates a scanner.
(2) SYNACK: number of TCP segments sent with both the SYN and ACK flag set. IP sources sending these packets tend to be servers. However, an abnormally high value may indicate a SYNACK attack.

(3) FIN & FINSENT: number of TCP FIN segments received/sent.
(4) RESET: number of TCP RESET segments received.
(5) TCPSENT & TCPRCV: number of TCP segments sent/received. This number is higher for servers as well as for p2p applications and is thus a useful discriminator.
(6) ICMP: number of ICMP errors received.

5.2 Composed attributes

We noticed that the simple (first order) attributes described above were not always able to distinguish between p2p applications and servers (e.g., TCPSENT and TCPRCV are high for both) or between p2p and scanners (e.g., if a p2p client has an out-of-date cache of peer IP addresses it may behave similar to a scanner). We developed the following composed attributes that are better at classifying traffic.

(1) WORK: the *work weight*, the ratio of TCP control packets to all TCP packets sent and received for an IP source. That is,

$$\frac{SYN + FIN + RESET}{TCPSENT + TCPRCV}$$

expressed as a percentage. 100% means more or less all control and no data. Obviously a web or ftp client system downloading a CD would tend to a work weight of 0. From experience we suggest that work weights above 50% should be deemed highly suspicious and are often worms or scanners. However there are rare cases of noisy clients that for some reason cannot reach their server or email servers that are trying to reply to spam (which will always fail). Low but non-zero work weights are often associated with p2p applications presumably because there are many TCP connections with less data download per connection when compared to conventional web and ftp downloads.

Although false positives are possible, we have observed the following about the WORK attribute:

- a system with a high work weight may very well be a syn scanner, or worm (a black hat flow), especially if there is a large number of syns per period, and very few or no fins.
- p2p applications like gnutella, kazaa, bittorrent, and edonkey/emule, sampled over millions of packets have average work weights as follows: gnutella: 30%, kazaa, 20%, bittorrent, and edonkey, less than 5%. It is possible that a badly behaved gnutella app in particular may have a high work weight (above 50%). This is because the gnutella application is failing to find any gnutella peers and thus behaves like a scanner. However p2p applications rarely score above 50%.

- Some layer 7 attacks will have lower work weights simply because 2-way data exchange is occurring. For example this is typical of password-based login attacks (with 1433 SQL servers as one form).
- In general work weights cluster around high values or low values. Values in the middle are less common.

(2) SAS: ratio of SYNACK segments to SYN segments sent by an IP source. That is,

$$\frac{SYNACK}{SYN}$$

expressed as a percentage. This measure is similar to the *functional role* attribute introduced in [7].

The SAS field expresses the total percent (0..100) of SYNACK packets typically sent as the second packet of the TCP 3-way initial handshake divided by the total number of SYN packets sent from the IP source in question. There are three possible thresholds here. 0 means the system in question is likely a client (or a scanner). 100 means the system in question is likely a server. A number in between (which is commonly found with p2p systems) shows that the system in question has both server and client functionality. One interesting facet of this field is that occasionally one will see a work weight of 100% and an SAS value of 100%. This means that the host in question is performing SYNACK scanning.

(3) L3COUNT: the number of unique destination IP addresses. Knowledge about how many different hosts a source communicates with is important in understanding its behavior in the network. This attribute is similar to the *popularity* explored in [7].

(4) L4COUNT: the number of unique destination ports.

(5) E : takes a value of *yes* or *no* and indicates *Errors* as a quadratic weight. The attribute is set to *yes* if $(SYN - FIN) * (ICMP + RESET) > 100000$. Intuitively, this flag indicates that the IP source was causing errors. Thus, this value is maximized if SYN > FIN and the sum of the number of resets and icmp errors is high.

(6) R : takes a value of *yes* or *no* and is set to *yes* if $(FIN == 0)AND(RESET > 5)$. This is an indication that an unexpectedly large number of resets are returned. This will occur if a host attempts to connect to ports that do not have any services listening.

(7) O : takes a value of *yes* or *no* and is set to *yes* if $(SYN > 20)AND(FIN < 5)$. It is an indication of too few (or none) FIN segments.

(8) M : takes a value of *yes* or *no* and is set to *yes* if no data TCP packets are returned to the IP source.

(9) W: takes a value of *w* if the work weight is between 50 - 90%, takes a value of *W* if the work weight is greater than 90%. Note that this attribute is highly correlated with the work weight and as a consequence was not used in our machine learning.

5.3 Rationale for various attributes

To explain the application of the attributes, we present summaries of data collected over 2000 30-second sample periods in Tables 2, 3, 4, and 5. The source IP addresses have been anonymized. *WORK* denotes work weight (the three values shown represent the min, average, and max) and *flags* denote which of the *E, R, O, M* attributes take a *yes* value. The *w* or *W* flag is set if the work weight attribute takes a value of between 50% – 90% or is above 90% respectively. The *apps* field indicates the application seen as determined by running our oracle (Figure 1).

IP_src	flags	apps	WORK (min/avg/max)	SAS	L3COUNT/ L4COUNT	SYN/FIN/RESET	TCPSENT/ TCPRCV
netW.1		H	0/0/3	100	22/53	40/38/0	20382/10723
netW.1	O	H	1/9/25	100	22/105	98/93/0	1105/880
netW.1	OR	H	0/3/36	99	10/14	10/7/0	936/837
netW.1		H	0/6/16	100	43/51	38/35/0	733/764
netW.1	w	H	0/8/50	100	19/164	159/155/0	2122/1777

H – web server based on source port (80/443)

Table 2
Campus web servers.

Table 2 shows the values taken by different attributes for web server traffic. The value of SAS tends to be close to 100% in all cases and this attribute is thus a good determinant of this type of traffic. On the other hand, SAS is low for email servers as shown in Table 3. This is because email servers can perform many email client connections to other email servers. In the case of both email and web servers, the work weight is typically small although there are periods with email servers where all connections will fail (due to spam or lack of availability of a peer).

IP_src	flags	apps	WORK (min/avg/max)	SAS	L3COUNT/ L4COUNT	SYN/FIN/RESET	TCPSENT/ TCPRCV
netE.1	WORM	IE	0/12/100	4	13/2	14/8/0	183/153
netE.2	WOM	IE	0/13/100	3	13/2	15/9/0	187/155

I – irc, E – email based on destination port

Table 3
Campus email servers.

Table 4 shows the signature left by scanners. The average work weight tends to be pretty high (between 76% and 94%). This is because there are a great many

syns being sent by the scanners but few packets are returned. The TCPSENT value is almost equal to the number of syns because each syn is counted in TCPSENT as well. The SAS attribute is zero because the attackers are only sending syns. We also note that in all the cases two or more flags from the set *E, W, O, R, M* are set which is another indicator of a malicious application.

IP_src	flags	apps	WORK (min/avg/max)	SAS	L3COUNT/ L4COUNT	SYN/FIN/RESET	TCPSENT/ TCPRCV
netS.2	EWOR	I	25/77/97	0	552/5	1101/25/7	1332/221
netS.3	EWORM	IH	44/94/100	0	643/2	1748/12/33	1838/107
netS.5	EWO	IH	20/88/95	0	1040/2	2181/28/33	2394/264
netS.6	WO	I	22/81/94	0	417/2	809/15/33	944/138

I – irc, H – web server based on source port (80/443)

Table 4
Scanners.

Table 5 shows a sample of data for p2p applications and irc. The p2p applications we show include bittorrent, morpheus, edonkey and gnutella. We observe that the work weight is small for p2p as well as irc. However, the SAS is variable for p2p while it remains small for irc. If we examine the values of work weight and SAS for irc and email servers, we see that both these values are small for both applications. However, if we also examine the values for TCPSENT/TCPRCV (i.e., number of TCP segments sent and received) we see a significant difference. The irc sources have very small values as compared to p2p hosts. This is because irc generates very little traffic in general.

Based on the discussion above, we can have a rough scheme for classifying traffic as follows:

	WORK	SAS	TCPSENT/TCPRCV
web server	small	large	very large
mail server	small	small	intermediate
scanners	large	small	large
irc	small	small	small
p2p	small	variable	large

However, on closer examination of the data, it is easy to see that using this table we will typically not be able to uniquely determine the type of application that is running in many cases. This is the reason to include additional attributes as those discussed previously. Indeed, since the relationship between attribute values and application are quite complex, we decided to use a

flags	apps	WORK	SAS	L3COUNT/ L4COUNT	TCPSENT/ TCPRCV
()	G	11	0	159/47	587/513
()	B	0	90	64/62	4136/291
()	B	0	100	86/84	42471/346
()	I	0	0	2/1	5/5
()	IH	18	0	2/7	16/28
()	B	0	20	120/119	4769/587
()	M	0	0	4/2	142/249
()	B	0	0	76/39	1044/101
()	G	22	0	49/24	326/292
()	e	9	21	119/59	632/546
()	GM	3	0	352/42	1732/264
I – irc B – bittorrent G – gnutella e – edonkey					
M – morpheus H – web server source port (80/443)					

Table 5
Predominantly p2p traffic.

Bayesian approach rather than a decision-tree based approach.

6 Evaluation of the algorithm

This section examines, in detail, the performance of the classifier that is produced using the approach outlined in Figure 1. The performance of the real-time version of the classifier is then studied in section 7. The specific questions we answer in this section are the following: what is the accuracy of classification? In the event that a IP source is misclassified, what are the possible reasons? How well or how poorly do different attributes perform in the classification task? Is there a difference between classifying local versus remote IP sources? And finally, but most importantly, how often do we need to retrain the classifier?

The method we adopt to answer the questions is to collect several data sets and use some of these for training and others for testing the classifier. Since our goal in this section is only to examine properties of the algorithm, we run the classifier on entire data sets rather than running it in real-time. Our data collection took place at two different times. In Jan 2005 on two different days, we collected 18 datasets consisting of 10 million packets each from the edge

of our campus network. Each capture took 5 minutes on average and analysis showed approximately 30000 unique IP sources communicating during this time. The datasets are labeled "A" to "R". In October 2005 we collected an additional 5 data sets labeled "S" to "W". These datasets can be obtained from [2]. The first 18 data sets were used for both training by the learner to produce the tables for the classifier as well as for testing the classifier. The same classifier (unchanged) was then used on the new 5 data sets and we note that *10 months after being trained, the classifier still approached very high accuracy in traffic classification.*

The data captured at the edge of the network enables us to view all traffic to and from our campus network. The data thus contains information profiling all internal hosts's Internet communication as well as external hosts communicating with hosts on our network. With the profiling of an IP source we can expect to capture the general behavior of internal IP sources with more accuracy than the external IP sources seen communicating with our hosts - simply because we are not seeing all the external hosts' communication and thus have more data about internal hosts than external hosts. With this reasoning it is expected that p2p classification will be more accurate for internal hosts than external hosts. This section examines this theory by first training and testing a solution of p2p detection by considering all IP sources in Section 6.1 and comparing these results with a solution that only considers local IP sources in 6.2.

Another topic we investigate is the quality of the attributes. We rely on a Naive Bayes learner and its operation can be damaged through the usage of redundant attributes which tend to add noise. In Section 6.3 we utilize a *wrapper* method with *greedy forward selection* [14, p.233] to search for relevant attributes and then re-test the performance of our algorithm using only these attributes. After the above studies, in Section 6.4 we examine the quality of our classification for different p2p applications. We observe that some p2p applications can be detected with higher probability than others and we speculate as to the possible reasons. Finally, in Section 6.5 we run the classifier on five new data sets collected ten months after the classifier was trained. We examine the accuracy of the classification as well as the *execution speed*. The conclusion is that the classifier is still very accurate and can run at a speed that allows near real-time classification.

6.1 All IP sources all attributes

We first explore the accuracy of our solution when it is presented with all (summarized) data available from the edge of the network. The training and testing data contains all attributes (except the "w/W" flags) described in Sec-

tion 5 for all IP sources in the datasets ("A" to "R"). In order to increase our confidence in the results obtained we used a method of *cross-validation* during the training and testing of our learner. We performed 18 experiments, each experiment used one dataset for testing and a concatenation of the remaining datasets for training. The results from these experiments are presented in Figure 2 with the x-axis showing which dataset was used for testing. Our experiments reported an average *probability of detection* of 64% and average *probability of false positive* of 20%. These values result in an average accuracy of 76.5% (see Table 1).

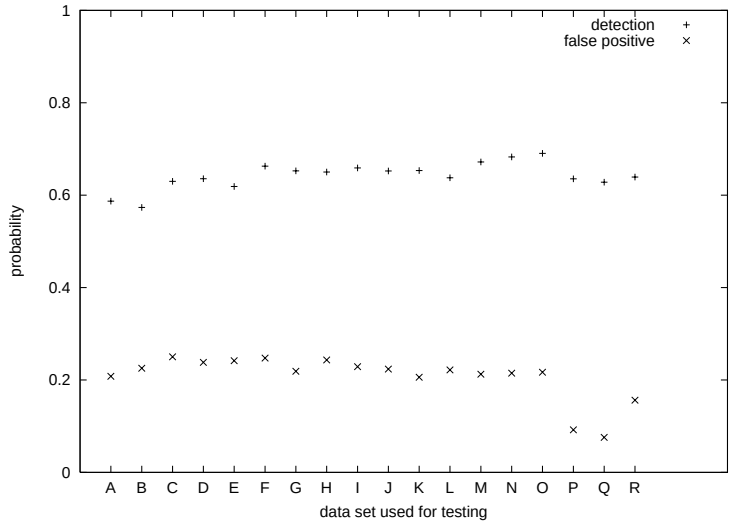

Fig. 2. p2p detection considering all IP sources

6.2 Local IP sources all attributes

The results from Section 6.1 show us that our suspicion of insufficient data for remote IP sources may be true. Our next experiment follows the same script as the previous, but only considered data for local IP sources (we discarded entries in the 18 datasets for non-local IP sources). Figure 3 presents the results from these experiments. The *probability of detection* improved significantly to an average of 76%, so did the *probability of false positives* (to an average of 12%). The average accuracy improved to 86.3%.

6.3 Attribute selection

Experience with the above experiments leads to the next conjecture that the accuracy can be improved by considering only the most relevant attributes rather than all the attributes. To test this conjecture, we implemented a *wrapper* method with *greedy forward selection* to eliminate redundant and

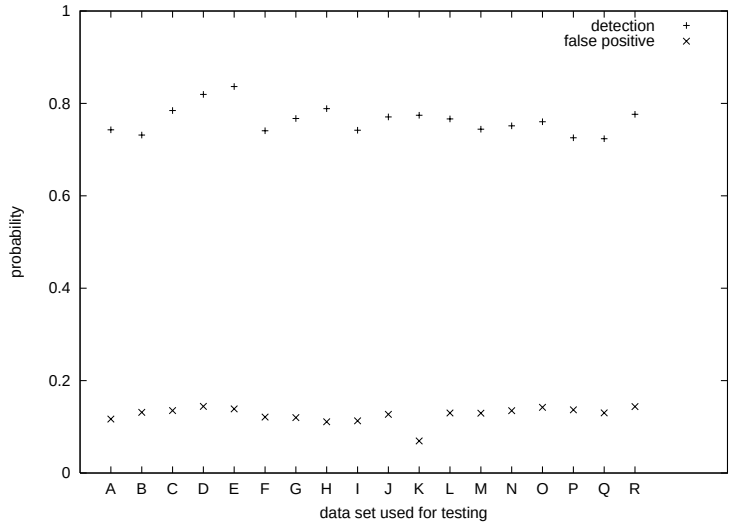

Fig. 3. p2p detection considering local IP sources

irrelevant attributes. That is, we start with no attributes and add them one at a time, each time presenting the new data to our solution of p2p detection. In each round the best performing attribute (based on *probability of detection* value) is carried over to the next round. In this way we iteratively find the best performing attribute, the best two performing attributes, the best three performing attributes, etc. To increase our confidence in our results we performed the attribute selection 18 times by creating the training and testing dataset in the same way as the experiments described in Sections 6.1 and 6.2. We tallied the order of attribute selection in each experiment and scored attributes to reflect how early they were selected by the method - lowest score means earlier selection. This new ranking across experiments provided us with the most relevant attributes. Table 6 presents the detailed results from our wrapper runs by showing the order in which the attributes were selected during each test run *for local IP sources only*. The table only shows the results for the top-5 attributes and the high numbers in the table thus show how many other attributes were selected before a particular attribute in a test run. It can be seen that the attribute selection showed significant variance during attribute selection (except for the top three attributes).

The overall results from the attribute selection are (from most relevant to least): TCPSENT, L4COUNT, TCPRCV, ICMP, WORK, FIN, R, M, FINSENT, SYN, E, O, SYNACK, RESET, L3COUNT, and SAS. A reflection on this selection does seem appropriate. We are searching for p2p applications that are typically used for large file transfers - an IP source typically receives and transmits large amounts of data during such a session. This supports the selection of the TCPSENT and TCPRCV attributes. p2p applications are also known to be unique in their connection patterns when examining the ratio of distinct IP addresses to distinct port numbers as these numbers tend to be equal, see [6] and [7], this property can be captured by L4COUNT. Hosts par-

Attribute	Dataset used for testing																		Average
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	
TCPSENT	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
L4COUNT	3	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2.06
TCPRCV	2	3	6	7	3	14	3	3	3	3	3	4	3	3	3	3	3	3	4
ICMP	5	6	10	8	10	8	6	7	6	7	5	8	8	9	9	4	6	8	7.2
WORK	4	13	13	15	15	6	11	5	5	6	11	6	6	6	5	9	4	4	8

Table 6
Wrapper attribute selection for local IP sources.

ticipating in p2p networks are also known to frequently join and disconnect from the p2p network, this will result in other hosts attempting to connect to them receiving failures and generally have an increased *work weight* as it searches for connected peers. Thus the usefulness of the ICMP and WORK attributes.

In order to see how the probability of detection changes with the use of different numbers of attributes, in Figure 4 we plot the probability of detection of p2p applications as a function of the attributes used for six representative data sets (A, D, E, K, O, P). We start with the most significant attribute TCPSENT and then add L4COUNT and then add TCPRECV, and so on. The data shows that there is little point in selecting more than the top five attributes as we see little change in the probability of detection. Table 7 shows the raw data used for the plot for one of the samples. We see that the probability of detection increases to 0.78 with the top five attributes and then it drops slightly as we add more attributes. Similar patterns are seen in other samples.

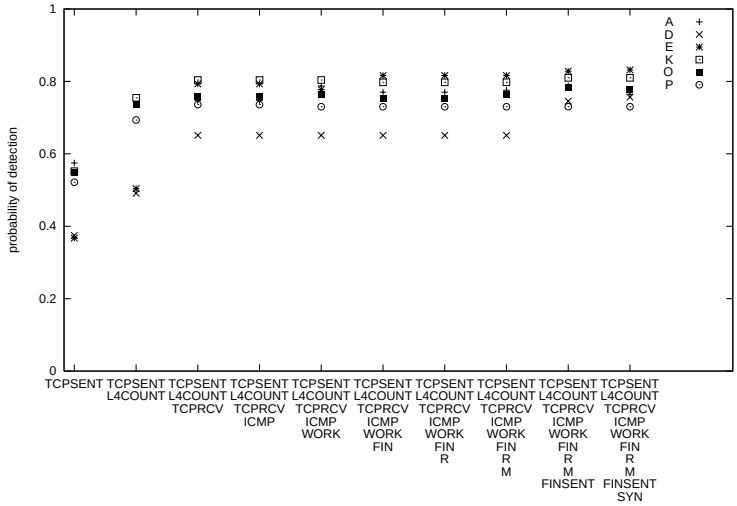

Fig. 4. Ranking of the attributes for local IP sources.

pd	*pf*	Attributes
0.574713	0.0343535	TCPSENT
0.735632	0.0718301	above + L4COUNT
0.747126	0.0730793	above + TCPRCV
0.747126	0.0730793	above + ICMP
0.787356	0.0780762	above + WORK
0.770115	0.0886946	above + FIN
0.770115	0.0886946	above + R
0.775862	0.0905684	above + M
0.787356	0.129919	above + FINSENT
0.764368	0.148032	above + SYN
0.764368	0.146159	above + E
0.764368	0.146159	above + O
0.764368	0.126796	above + SYNACK
0.758621	0.123673	above + RESET
0.764368	0.136165	above + L3COUNT
0.747126	0.117427	all attributes

Table 7
Change in probability of detection as we use more attributes for one data sample (A) for local IP sources only.

6.3.1 Using only the Top-5 attributes

We next explore how well the top attributes do in detecting the presence of p2p applications. We first repeat the experiments from Section 6.2 but only use the top-5 attributes: TCPSENT, L4COUNT, TCPRCV, ICMP, and WORK. The results from these experiments are presented in Figure 5 for local IP sources. As is clear, the usage of the top five attributes significantly improves the *probability of false positives*, the average is now a very low 7.8%. The average *probability of detection* did drop slightly to 75%. The average accuracy however improved significantly to 90.4%

We next repeat the experiments of Section 6.1 (i.e., all IP sources) but only using the top five attributes as above. Unlike the case with local IP sources, the probability of detection decreases from 64% (using all attributes) to only 48% (using the five attributes from above). However, the probability of false positives also decreases from 20% to 14.7% which actually improves the overall accuracy from 76.5% to 77.98%. Figure 6 plots the probability of detection and probability of false positives for this case. Even though the accuracy has

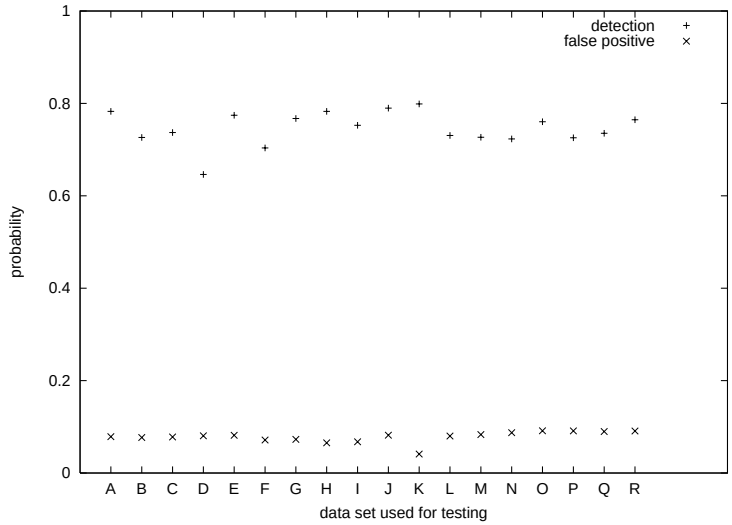

Fig. 5. p2p detection considering local IP sources - top-5 attributes

improved, we believe that using the top-5 attributes selected by applying the wrapper method to local IP source data only is problematic. Thus, we now repeat the wrapper algorithm for all 18 data sets considering all IP sources.

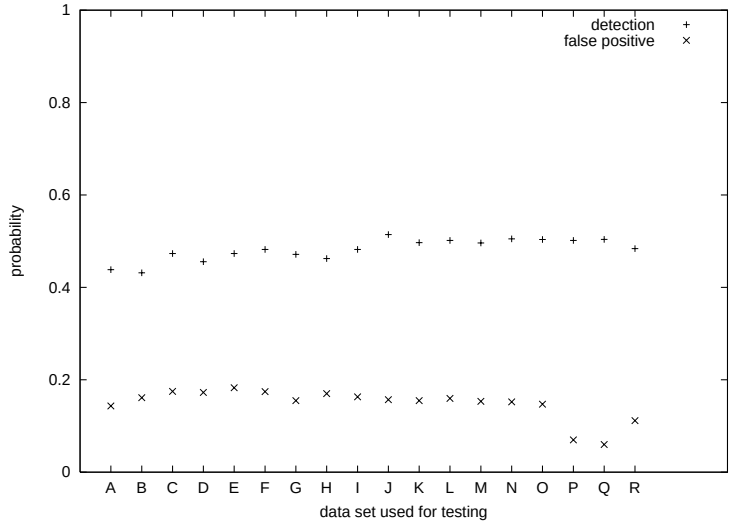

Fig. 6. p2p detection considering all IP sources - top-5 attributes

6.3.2 All IP sources top-8 attributes

When we run the wrapper algorithm considering all IP sources, we get a different ranking of the top attributes as indicated in Table 8 (to save space, this table only contains data corresponding to the first and last columns of the similar Table 6). Unlike in the case with only local IP sources, the work attribute is the most significant. Figure 7 plots the probability of detection of p2p applications as a function of the attributes (as was done for Figure 4). We see that the probability of detection continues to increase until we pick 8

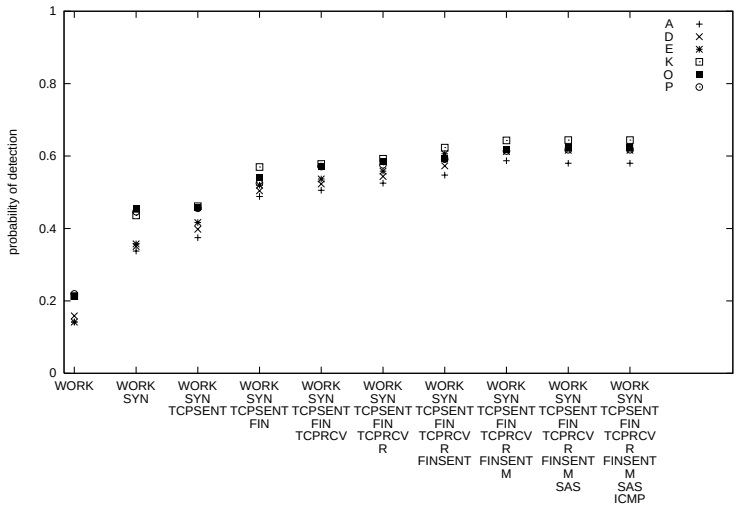

Fig. 7. Ranking of the attributes for all IP sources.

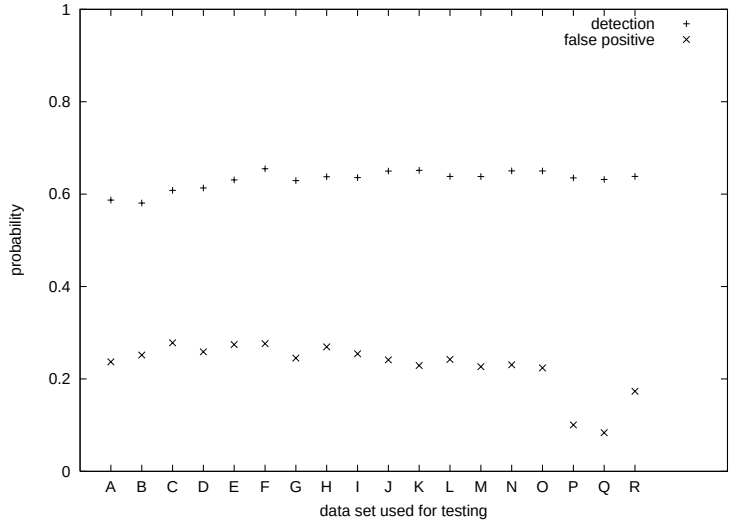

Fig. 8. p2p detection for all IP sources using top-8 attributes.

attributes after which it remains fairly constant. Therefore, for the case when we look at all IP sources, we consider the top-8 attributes and the results we obtain are as follows (see Figure 8): the average probability of detection is now 63% with a probability of false positives of 22% resulting in an overall accuracy of 75%. These numbers are almost identical to the case when we use all the 16 attributes (Section 6.1) which means that we can improve the speed of classification (using fewer attributes) without affecting accuracy. Table 9 summarizes the performance of our algorithm on the 18 datasets. In general we see that accuracy is better than 75% for all IP sources and better than 85% for local IP sources. The table also shows the accuracy for various selections of top attributes.

As with the previous attribute selection we consider the knowledge gained from the attribute selection. Of the top-5 attributes selected using datasets only

Attribute	Average
WORK	1
SYN	2.82
TCPSENT	4.36
FIN	5
TCPRCV	5.09
R	5.73
FINSENT	6.36
M	7.73

Table 8
Wrapper attribute selection for all IP sources.

containing local IP sources we note that L4COUNT stands out as not being as significant during attribute selection when using all IP sources. This is intuitive when viewed in the context of the amount of information we have about IP sources. We have full information about local IP sources participating in p2p networks. Connections from these hosts are to several destinations (local and remote) and each connection typically occurs to a distinct destination port. This connectivity is captured by the value of L4COUNT. When we are dealing with a remote IP source that participates in a p2p network then we cannot view its full connectivity to other p2p peers (assuming that most of its peers are not located in our network) and the L4COUNT is thus not as significant. When dealing with remote IP sources that participate in p2p networks the top-8 attributes appear to capture two distinct cases: when a remote IP source is able to find a p2p peer in our network or not. The significance of SYN, TCPSENT, FIN, TCPRCV, and FINSENT match well to successful connections (and large data transfers) between local and remote p2p network peers. The significance of WORK, R, and M indicates the error rates experienced by remote IP sources as they attempt (and fail) to connect to local p2p hosts. The remote hosts failed attempts to connect is apparent through the significance of the amount of control data they send (WORK) and how the local IP sources respond by sending resets or not answering at all - as is evident through the significance of the R and M flags.

6.4 Detecting specific p2p applications

In this section we ask the following question, 'given a particular p2p application, how well does our method perform at detecting it?'. The results in the previous sections simply lumped together the probability of detection for all p2p applications. However, if some applications are easier to detect and occur

Experiment	*pd*	*pf*	**average accuracy**
All IP sources, all attributes	64%	20%	**76.5%**
Local IP sources, all attributes	76%	12%	**86.3%**
Wrapper method applied to local IP sources only			
All IP sources, top-5 attributes	48.2%	14.7%	**77.98%**
Local IP sources, top-5 attributes	75%	7.8%	**90.4%**
Wrapper method applied to all IP sources			
All IP sources, top-8 attributes	63%	22%	**75%**

Table 9
Summary of p2p classification performance for the original 18 datasets

more often then this probability will be higher as compared to the case when the more frequent p2p applications are harder to detect. Table 10 presents the detection behavior of our approach considering only local IP sources. Recall from Table 1 that d indicates that the oracle and our detector agree that the application is p2p whereas b indicates the case when the oracle says p2p but our detector misclassifies the source. The last column in Table 10 is the probability of detection computed as the ratio $\frac{d}{b+d}$. The last row of the table gives us a probability of detection of 80% while the results from Section 6.3 reported a probability of detection of 75%. The difference in these two numbers comes about as follows. The 75% value resulted from a d value of 2499 and a b value of 845 giving us $PD = \frac{2499}{2499+845} = 75\%$. The training and testing data used in Section 6.3 *only considers the classification of a host*: whether it is running a p2p application or not. In our initial classification by the oracles we found that *hosts very often run more than one p2p application*. For the previous experiments this was not a concern as these cases will cause a host to carry the encompassing classification of "p2p". Multiple p2p applications on a host does become important now that we are considering individual p2p applications. A close look at the data from Section 6.3 (3344 hosts running p2p applications) revealed 4483 instances of p2p applications, thus there were 1139 cases where more than one p2p application was running on a host. Completing the confusion matrix for these tests resulted in $d = 3601$ and $b = 882$. For individual p2p applications we thus have a *probability of detection* of 80%.

Returning to Table 10 we note that p2p applications such as bittorrent and directconnect have a almost 100% probability of detection while edonkey has a value of only 74%. It is interesting to examine why edonkey has a lower probability of detection. Figure 9 plots the value of the attribute L4COUNT as a function of a count of instances[4]. From the point of view of using the

[4] In other words, a value (x, y) says that there are x instances when the oracle flagged p2p for which the value of L4COUNT is y.

L4COUNT as an attribute for classifying p2p applications, the problem is evident – this attribute is a poor discriminator for edonkey since the values for edonkey are spread out widely whereas all the other p2p applications have values clustered towards the lower values of the x-axis. On the other hand, we see that the values of this attribute for bittorrent are clustered towards the lower values of the x-axis as well (the values marked 'o'). The large number of values marked '+' that are spreadout correspond to edonkey (since '+' plots all p2p except bittorrent). We note that very similar graphs are produced for the other four top-5 attributes.

In examining the raw data in more detail, we observe that the oracle frequently classifies edonkey with one or more other p2p application. Table 11 provides the number of times a given p2p application was classified with another p2p application for one data set only considering local IP sources. Because of the large number of times edonkey was classified with another p2p application, we believe that the oracle has difficulty discriminating between edonkey and other p2p applications. This belief is supported when we compare the regular expressions implemented in the l7-filter to detect edonkey and bittorrent [5]

bittorrent pattern:

```
^\x13bittorrent protocol
```

edonkey pattern:

```
^[\xe3\xc5\xe5\xd4].?.?.?.?([\x01\x02\x05\x14
\x15\x16\x18\x19\x1a\x1b\x1c\x20\x21\x32\x33
\x34\x35\x36\x38\x40\x41\x42\x43\x46\x47\x48
\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53
\x54\x55\x56\x57\x58\x5b\x5c\x60\x81\x82\x90
\x91\x93\x96\x97\x98\x99\x9a\x9b\x9c\x9e\xa0
\xa1\xa2\xa3\xa4]|\x59..............?[ -~]
|\x96....$)
```

Indeed, it is likely that conditions which make the edonkey expression true have a high probability of making another regular expression true as well. We therefore believe that the oracle frequently overestimates the number of edonkey instances which results in our learner being misinformed which in turn leads to a somewhat lower probability of detection of all p2p instances. Put another way, our classifier is underperforming because the oracle is not entirely accurate in identifying edonkey (though it is very accurate in detecting the other applications). Finally, we note that the difficulty of properly classifying

[5] \x <value> – refers to a hex number in *value*, the caret character refers to the start of a line, [...] indicates a class and the pattern matches any of the members, ? means match at most once, and a period means any character.

p2p application	b	d	PD
100bao,applejuice,ares gnucleuslan,goboogy,hotline napster,poco,soribada,tesla	0	0	0
bittorrent	13	884	99%
directconnect	0	21	100%
edonkey	736	2045	74%
fasttrack (includes kazaa and morpheus)	4	14	78%
gnutella	15	115	88%
kugoo	5	36	88%
mute	16	60	79%
openft	0	3	100%
soulseek	93	423	82%
All	882	3601	80%

Table 10
Detection of individual p2p applications

p2p application	Multiple count
eDonkey	798
Bittorrent	674
Directconnect	9
Fasttrack	12
Gnutella	85
Kugoo	40
Mute	59
Openfit	3
Soulseek	366

Table 11
Times a p2p application was classified with another.

edonkey has also been discussed elsewhere [13].

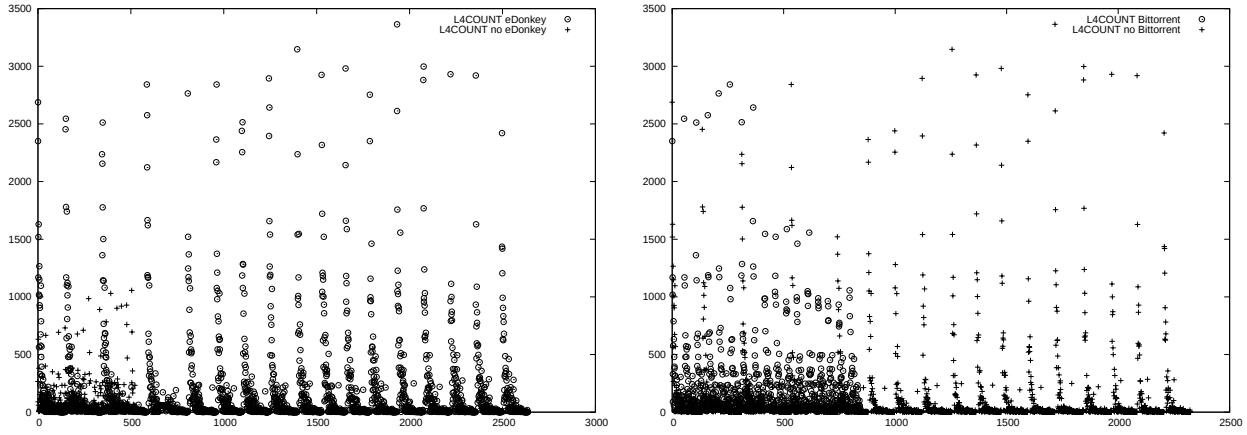

Fig. 9. Behavior of edonkey and bittorrent on L4COUNT.

Experiment	*pd*	*pf*	**average accuracy**	Instances	Total Time
All IP sources, all attributes	55%	29%	**68%**	40771	23.7s
All IP sources, top-8 attributes	55%	32%	**65%**	40771	12.6s
Local IP sources, all attributes	81.5%	10.6%	**88.8%**	2370	1.43s
Local IP sources, top-5 attributes	79.5%	5.8%	**93%**	2370	0.5s

Table 12
Measures of p2p classification performance for the 5 new datasets

6.5 Running the classifier on new data

The original 18 traces were collected in January 2005 which were used to train the classifier. We collected five new data sets in October 2005 and ran the classifier again (but we did not train the classifier on this new data). The results of the classification are very encouraging and are summarized in Table 12. The main conclusions are (1) the classifier works as well as it did ten months ago which means that frequent training is unimportant (2) the accuracy of detecting local IP sources improves significantly using only the top-5 attributes while it decreases a bit when using top-8 attributes for all IP sources (compared to Table 9), and (3) the benefit of using fewer attributes is the lower processing time – thus, the slight lowering of accuracy for a 2x reduction in processing time for all IP sources is a very productive tradeoff.

The classification performance was measured on a 2GHz Intel Pentium machine with 512MB RAM running Linux kernel 2.6.11. The classifier was implemented as a script in gawk. The classifier iterated ten times over the testing dataset performing the entire classification process every time and the values in the table indicates the average speed.

7 Real-time P2P detection

The experiments described in the previous section performed *offline* P2P detection since our goal was to understand the algorithm in detail. We now consider a real-time implementation of the classifier. Unlike signature based classifiers, our classifier uses statistical attributes to classify IP sources. Therefore, the question arises, how can we classify an IP source in real-time since we need statistical data?

The abstract view of the approach we use is to keep a T second *history* of traffic already seen and use this history to compute attribute values for each IP source. Then when we see a next occurrence of such an IP source, we can update the attribute values and classify it immediately. However, a problem with this approach is that keeping and using T seconds worth of traffic will make the implementation very slow. The approach we adopt is therefore to use *attribute summaries* of T seconds worth of traffic and to update these summaries as more traffic is collected. To be clear, these attribute summaries are computed for the data that is there in the sliding window of length T only.

Our implementation of the real-time classifier is constrained by the frontend (see Figure 1) which collects and summarizes attributes in 30 second chunks. Thus, T is a multiple of 30 seconds in our implementation. Let us say $T = 30k$ seconds. Then for each of the latest k 30 second samples provided by the frontend, we compute the values of the top-5 attributes for local sources (and top-8 for remote sources). However, the classifier uses a *normalized* frequency table to do the classification. Recall that the classifier was trained on 10 Million packets and to use it on smaller samples, we need to normalize it. We maintain state in the classifier. For each IP source we maintain a cumulative count of the values for its top 5 attributes in window T. When the *classifier* is presented with a new 30 second sample it adds the values of the current top attributes to those recorded from previous samples and then *normalizes* these over the total number of 30 second samples seen up to that point. We measured the performance of our classifier when the summarizer is presented with the 30 second periods of each of the five traffic captures from October 2005.

Figure 10 shows the performance of the classifier for different values of k. On the x-axis (labeled 'sample'), a value of, say 6, means that we used $T = 6 \times 30 = 180$ seconds. The idea here is to see what value of k represents a tradeoff between speed and accuracy. From this graph (as well as others created for the other data sets), it appears that $k = 5$ works well in terms of maintaining little state (i.e., attribute values) and high accuracy. From this we see that the *probability of detection* and *probability of false positive* reach a good balance after seeing about five 30 second samples.

Figures 11 presents the pd results from these experiments for each of the five traffic captures in October 2005 (captures named "S" to "W") using $T = 5 \times 30 = 150$ seconds. Note that the x-axis starts at 5 because of this selection of T. We also note that the *probability of false positive* values were 0.14. The *probability of detection* values over two samples, "U" and "V", do seem lower than the average but we observed that this is not due to the usage of the sliding window. Indeed, this appears to be an artifact of the data sets themselves.

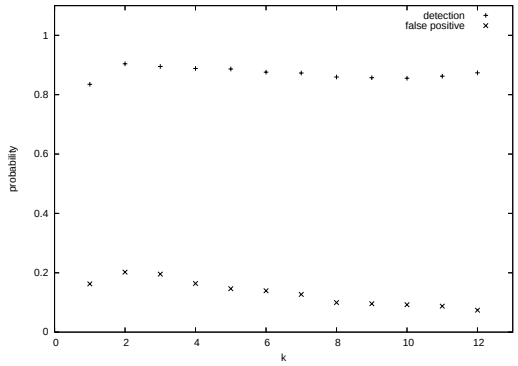

Fig. 10. Real-time p2p detection using 30s samples of traffic capture "S" cumulatively

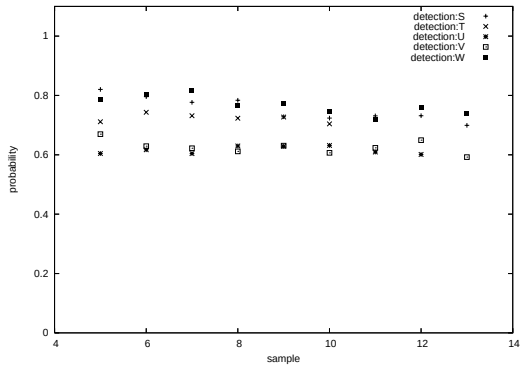

Fig. 11. p2p detection using sliding window over 30s samples

8 Encrypted p2p traffic

In the approach described thus far, the p2p traffic needs to be unencrypted in order for the oracle to appropriately train the classifier. This presents a problem for the future when the expectation is that all p2p traffic will be encrypted. In order to circumvent this problem, we propose the following modification to the approach from Figure 1. We introduce a host into the local network and configure it to run one p2p application and set it running. Data is collected as described previously for some length of time. Of the data collected, all the encrypted traffic is discarded with the exception of the encrypted traffic bearing our host's IP address in the IP header. This process is repeated for

all known p2p applications and at the end we concatenate all these doctored traffic traces together and this now forms the training set. This technique will thus allow our approach to be used in the future as well. There is however one caveat: there will be encrypted traffic other than p2p traffic that we are discarding. It is hard to predict how this will affect the classifier's performance since we have no actual figures available today that quantify the percentage of this form of traffic. However, this is an important research problem that needs to be addressed in the near future.

9 Discussion and Future work

In this work we have shown that it is possible to detect the presence of p2p traffic without considering the payload of packets. Tables 9,12 present a summary of our results. As far as we know this is the highest accuracy, highest *probability of detection*, and lowest *probability of false positive* with which p2p detection have been accomplished in this domain. The very low *probability of false positives* makes this solution particularly attractive to network operators who rely on these measures for visibility into their network usage. Another important contribution of this paper is the definition of a rigorous experimental method that facilitates comparison between new and prior results. We report accuracy, PD, and PF in cross-validation across multiple public domain data sets and urge other researchers to do the same. Finally, we have produced a real-time classifier for p2p traffic that does not require frequent retraining. Indeed, as our data shows, the classifier continued to have high accuracy 10 months after being trained.

References

[1] l7-filter web page, 2005. URL: http://l7-filter.sourceforge.net.

[2] p2p classification dataset, 2005. URL: http://unbox.org/data/Paper1568975554.zip.

[3] http:// torrentfreak.com/ encrypting-bittorrent-to-take-out-traffic-shapers/, 2006.

[4] William Aiello, Charles Kalmanek, Patrick McDaniel, Subhabrata Sen, Oliver Spatscheck, and Jacobus Van der Merwe. Analysis of communities of interest in data networks. In *Passive and Active Network Measurement: 6th International Workshop, PAM 2005, Boston, MA, USA, March 31 - April 1, 2005.*

[5] Felix Hernandez-Campos, F. Donelson Smith, Kevin Jeffay, and Andrew B. Nobel. Statistical clustering of internet communication patterns. In *Proceedings*

35th Symposium on the Interface of Computing Science and Statistics, July 2003.

[6] Thomas Karagiannis, Andre Broido, Michalis Faloutsos, and Kc claffy. Transport layer identification of p2p traffic. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 121–134, 2004.

[7] Thomas Karagiannis, Dina Papagiannaki, and Michalis Faloutsos. Blinc: Multilevel traffic classification in the dark. In *ACM SIGCOMM*, 2005.

[8] Andrew W. Moore and Konstantina Papagiannaki. Toward the accurate identification of network applications. In *Proceedings of Sixth Passive and Active Measurement Workshop (PAM 2005)*, 2005.

[9] Andrew W. Moore and Denis Zuev. Internet traffic classification using bayesian analysis techniques. In *Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 50–60, 2005.

[10] Matthew Roughan, Subhabrata Sen, Oliver Spatscheck, and Nick Duffield. Class-of-service mapping for qos: a statistical signature-based approach to ip traffic classification. In *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 135–148, 2004.

[11] Subhabrata Sen, Oliver Spatscheck, and Dongmei Wang. Accurate, scalable in-network identification of p2p traffic using application signatures. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 512–521, 2004.

[12] Subhabrata Sen and Jia Wang. Analyzing peer-to-peer traffic across large networks. *IEEE/ACM Trans. Netw.*, 12(2):219–232, 2004.

[13] Kurt Tutschku. A measurement-based traffic profile of the edonkey filesharing service. In *5th Passive and Active Measurement Workshop PAM2004, France, 2004*.

[14] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999.