Evaluating Machine Learning Methods for Online Game Traffic Identification

Nigel Williams, Sebastian Zander, Grenville Armitage
Centre for Advanced Internet Architectures (CAIA). Technical Report 060410C
Swinburne University of Technology
Melbourne, Australia
{niwilliams,szander,garmitage}@swin.edu.au

Abstract—**Online gaming is becoming more and more prominent in the Internet, in terms of both traffic volume and as a potential source of revenue. Quality of Service (QoS) requirements for highly interactive games are much stricter than for traditional Internet applications, such as web or email. For effective QoS implementations that are transparent to users and game applications, an accurate and reliable method of classifying game traffic flows in the network must be found. Current methods such as port number and payload-based identification exhibit a number of shortfalls. A potential solution is the use of Machine Learning techniques to identify game traffic based on payload independent statistical features such as packet length distributions. In this paper we evaluate the effectiveness of the proposed approach. We compare the accuracy and performance of different Machine Learning techniques and we also use feature selection techniques to examine which features are most important in discriminating game traffic from other traffic. We find that machine learning algorithms are able to separate online game traffic from other network traffic with very high (>99%) accuracy. We also show that feature selection, while reducing accuracy, allows games to be identified with fewer features and substantial speed gains.**

Keywords—**Game Traffic Classification, Machine Learning, Statistical Features**

I. INTRODUCTION

The Internet is experiencing an increase in the use and commercialisation of interactive applications such as telephony and online gaming. Online gaming in particular is expected to become a large source of income, through either subscription-based games or dedicated gaming services. Internet Service Providers may also charge a premium for Quality of Service (QoS)-enhanced accounts targeted at gamers.

Highly interactive online games, such as First Person Shooter (FPS) games, have a narrow tolerance to network issues such as delay, jitter and packet loss (see [1], [2]) necessitating more rigid QoS compared to the best effort service used for traditional Internet applications such as web or email. In order for QoS to be effective however, an accurate and timely method of identifying and classifying network gaming flows is required. As it is unlikely that game applications will ever explicitly signal their QoS demands to the network, the network must identify game flows and establish adequate QoS for these flows. Once highly interactive game traffic can be identified it can be given a higher priority over other traffic in the network. We presented the architecture and advantages of such a system in [3].

Current popular methods of classifying network applications include TCP/UDP port-based identification, and payload-based identification. The latter can be further divided into protocol decoding and signature-based identification. With protocol decoding the classifier actually decodes the application protocol while signature-based methods search for application specific byte sequences in the payload.

Port-based classification systems are moderately accurate at best and will become less effective in the near future. For example, a server hosting multiple games or instances of the same game might use an arbitrary port rather than the specified default port, making port-to-application mappings unpredictable. Payload-based classification relies on specific application data, making it difficult to detect a wide range of applications or stay up to date with new applications. In addition, the process of creating rules for signature-based classification must often be done by hand, which can be very time consuming.

Machine learning (ML) techniques [5] provide a promising alternative through classifying flows based on application protocol (payload) independent statistical features. The features used in this study are flow characteristics such as packet length and inter-arrival times. This approach does not require packet payload and the classifier can be trained automatically assuming a representative training dataset can be obtained. A more general introduction to the problem is presented in [6] and [7].

We have previously used a wide range of machine learning algorithms to separate common network applications, such as web and mail traffic [24]. In this paper we apply several of the better performing algorithms to the task of separating network games from generic (i.e. common) network traffic. Although this is not the main focus we also investigate how effectively different games can be separated from each other. As the

features used in classification have a crucial impact on accuracy we also evaluate and compare performance of the algorithms using Correlation-Based Feature Selection (CFS).

For our evaluation we use gaming data captured by members of CAIA [8] and Mark Claypool [9]. 'Generic' traffic examples were taken from several publicly available traffic traces. We predominantly focus on First Person Shooter (FPS) games as these fast-paced games have the most stringent QoS requirements. However, we have also included some data of a Real Time Strategy (RTS) game. The games tested were from the PC and Xbox platforms. It is important to include Xbox traffic as current and next-generation console devices such as the Xbox 360 and PlayStation 3 are expected to produce a significant share of online gaming traffic [10].

We find that some algorithms are able to separate the different games from each other and other traffic with very high (>99%) accuracy. Results were also encouraging when using a reduced number of features (after feature selection). However, while the overall accuracy of separating game from non-game traffic is similar, the accuracy for detecting some individual games is poor.

We found that all of the ML techniques seem to be fast enough for real-time classification of a fairly large number of simultaneous flows (at least several thousands per second). Furthermore, most of the algorithms can train fast enough to allow for frequent updates of the classifier (training took no more than half an hour).

The paper is structured as follows. Section 2 discusses the shortfalls of current classification approaches and outlines our machine learning based approach. Section 3 describes related work. Section 4 and 5 describe the feature selection and ML techniques we use. Section 6 describes our datasets and approach. Section 7 presents the results of the evaluation and Section 8 concludes and outlines future work.

II. Current Classification Techniques

A. *Port Numbers*

The oldest and still most common technique is based on the inspection of known port numbers. While some applications use symmetric ports (all communicating peers use the same port number), many client-server applications such as the web are port asymmetric (only the server is using the well-known port whereas clients use dynamic ports). Therefore in this paper we refer to either port numbers or the server port as the port that identifies an application.

The Internet Assigned Numbers Authority (IANA) [11] assigns the well-known ports from 0-1023 and registers port numbers in the range from 1024-49151. Many applications do not have IANA assigned or registered ports however and only utilise 'well known' default ports. Often these ports overlap with IANA ports and an unambiguous identification is no longer possible. A port database [12] that lists not only the IANA ports but also ports reported by users for different applications shows that many applications have overlapping ports in the IANA registered port range. As more and more applications emerge, this overlap will increase since the port number range is not likely to increase.

Port-space overlap is also caused by users running applications with well-known ports on arbitrary ports to bypass port-based filters or hide traffic. Furthermore applications such as passive FTP and video/voice communication choose ports dynamically.

In general online games have only a commonly known default port. As these ports are not IANA registered, there is potential for other applications to also use the same port. Often online gaming servers run multiple servers on a single physical host (IP address), which means every server must run on a different port and therefore many servers run on non-default ports.

B. *Protocol Decoding*

A more reliable technique used in many current industry products involves stateful reconstruction of session and application information from packet content (e.g.[13]). This technique avoids reliance on fixed port numbers and provides very accurate and reliable application identification, but imposes significant complexity and processing load on the traffic identification device. It must be kept up-to-date with extensive knowledge of application semantics and network-level syntax, and must be powerful enough to perform concurrent analysis of a potentially large number of flows.

This approach can be difficult or impossible when dealing with proprietary protocols or encrypted traffic. Another problem is that direct analysis of session and application layer content may represent an explicit breach of organisational privacy policies or violation of relevant privacy legislation.

This method currently provides the highest accuracy and reliability for classifying network traffic to corresponding applications. The major problems of this method are the performance required (especially considering the ever-increasing network bandwidths) and the effort required for implementing and keeping the protocol definitions up to date. In our opinion this method seems only feasible for few applications when the incentives to provide reliable classification are very high.

As many game protocols are not openly specified (to prevent players from cheating), protocol definitions for online games can only be obtained through reverse engineering (not always a straightforward task).

C. *Signature-based Approaches*

To overcome the inefficiencies of protocol decoding some researchers have proposed the use of signature-based methods. These methods search packet payload for the characteristic patterns of specific applications. The advantage of this approach is that it can be more effective than pure port-based classifications and more efficient than protocol decoding (though less accurate). Overall, signature-based methods provide a very good trade-off between resource efficiency and classification performance.

This method is still protocol dependant however, and as with payload inspection signatures must be developed using protocol specifications or through reverse engineering. There are also a number of ways to defeat simple signature-based detection (e.g. [43]). Signatures cannot be used on encrypted data.

This approach seems feasible for online games but very cumbersome for the scenario in which real-time interactive flows need to be separate from non-real-time flows, as signatures would need to be developed for every game (or alternatively for every non-game).

III. ML-BASED CLASSIFICATION

A. *Approach*

Figure 1 visualises a machine learning based classification architecture. Training input data can be taken from previously captured traffic traces (or possibly from live capturing). Then packets are grouped into flows based on IP addresses, TCP or UDP ports and protocol and the flow characteristics (features) are computed. The flow data used for training each class must be representative for the particular network application. For supervised learning algorithms the flow data needs to be labelled with class labels corresponding to the network applications prior to training. For large data traces it is necessary to limit the number of flows passed to the learning algorithm by sampling flows before training.

The flow characteristics and a set of algorithm parameters are then used to build a classification model (see Figure 1 top). The algorithm parameters range from very simple to very complex and depend on the ML algorithm used. For some algorithms no parameters may be needed. Once the classifier has been trained new flows can be classified based on their statistical attributes (see Figure 1 bottom). New flows are taken from live network capture or from trace files. Again sampling can be used to only classify a fraction of the overall flow data for example if the classification performance is insufficient. The results of the classification process can be used to map network traffic to different QoS classes, or other tasks such as trend analysis.

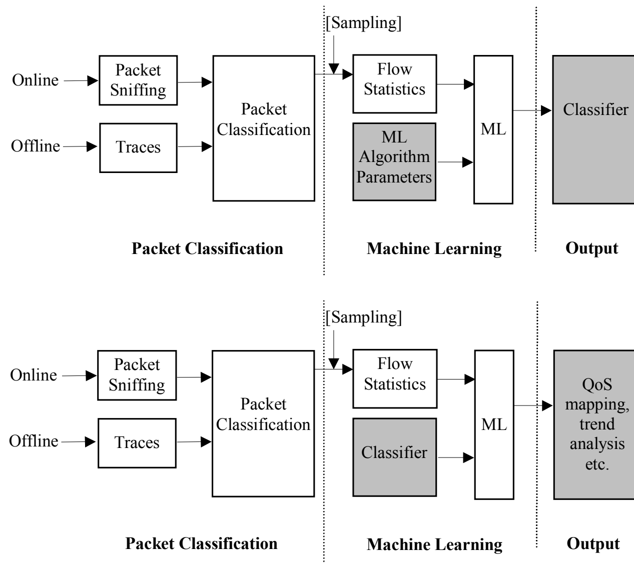

Figure 1: ML-based flow classification approach: learning phase (top) and classification phase (bottom)

B. *Assessing Performance*

There are several approaches to testing the accuracy of ML algorithms. We use the common method of k-fold cross validation. In this process the data set is divided into k subsets. Each time, one of the k subsets is used as the test set and the other k-1 subsets are put together to form the training set. Error statistics are calculated as the average across all k trials. This allows an overall indication of how well the classifier will perform on unseen data.

We use three standard metrics to evaluate the performance:

1. Accuracy is the percentage of correctly classified instances over the total number of instances.
2. Precision is the number of class members classified correctly over the total number of instances classified as class members.
3. Recall (or true positive rate) is the number of class members classified correctly over the total number of class members.

A confusion matrix provides the basis for the evaluation metrics and contains information about actual and predicted classifications for each class. Table 1 shows the confusion matrix for a two-class classifier (classes - and +).

Table 1: Confusion matrix

Label *l*	**Assignment *z***	
	-	+
-	**TN**	**FP**
+	**FN**	**TP**

Each instance is associated with a label *l*, which accounts for the true class. The classification produces an assignment *z* 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.

Accuracy is computed with the following formula:

$$a = \frac{TP + TN}{FP + TP + FN + TN} \quad (9)$$

Precision is calculated using the following formula:

$$p = \frac{TP}{TP + FP} \quad (10)$$

The following formula is used for recall:

$$r = \frac{TP}{TP + FN} \quad (11)$$

In addition to the standard overall accuracy, precision and recall metric the processing performance of the algorithms is also evaluated. Two performance metrics were measured, defined below:

Classifications per second: The number of instances that the algorithm is able to classify each second. Speed is important to perform near real-time classification on large numbers of simultaneous networks flows.

Build Time: The time required to build a classification model. Building the classifier can be done offline but as building times may reach several days for certain classifiers, shorter build times may be more convenient.

IV. Related Work

The idea of using ML techniques for flow classification was initially introduced in the context of intrusion detection [14]. The authors evaluated how likely an identified flow was an intrusion based on the flow attributes of duration, number of packets and bytes using a decision tree as classifier. Although this work is not directly related to ours it is one of the earliest examples of using ML techniques in the area of computer networks. Since then many researchers have investigated the use of various ML methods for intrusion detection.

The authors of [15] use Principal Component Analysis (PCA) and density estimation to classify traffic into different applications. They use the distributions of two flow attributes: packet length and packet inter-arrival time. First they project each packet into a 2D bin matrix based on its length, the inter-arrival time and the direction of the packet and the previous packet. They then use PCA to reduce the high dimensional space and binning density estimation to learn each flow type from three PCA. The classification works in a similar way. After PCA the probability of a flow being an application is computed for the learned density of all applications and the application with the highest probability is select. The evaluation is based on a set of roughly 17,000 flows from two hours, ignoring flows with less than 10 packets. The false negative error rates are between 0% and 7.5%. The authors study traffic of the following well-known ports: FTP, SSH, RSH, telnet, DNS, SMTP, rlogin, ICMP and IRC.

In [16] the authors use nearest neighbour (NN) and linear discriminate analysis (LDA) to successfully map different applications to different QoS classes using four different attributes: average packet size, root mean square packet size, average duration, inter-arrival variability. The paper investigates the following applications: DNS, FTP data, HTTP over SSL, Kazaa, Realmedia and telnet. The authors do not use web traffic for the learning but show that web traffic falls into different classes and is difficult to separate from several of the other applications. The classification errors reported are between 2% and 13%. The flow attributes used in the study have not been computed based on single flows but are the mean values of all flows of a particular application aggregated over 24 hour periods.

The Expectation Maximization (EM) algorithm is used in [17] to cluster flows into different application types using a fixed set of attributes. The algorithm was found to separate traffic into a small number of basic classes. Cross validation was used and identified similar sets of classes across different parts of the same trace and different traces. However, from their evaluation it is not clear what influence different attributes and EM parameters have. Also, as the results of the clustering were not evaluated it is unclear as to how good the clustering was.

In [18] the authors use a simulated annealing EM algorithm to classify traffic flows based on their size (e.g. mice and elephants). The authors conclude that their approach produces more meaningful results than previous threshold-based methods.

We proposed an ML-based approach for identifying different network applications in [6] using unsupervised learning. This approach is based on the autoclass [19] algorithm, which is based on Bayesian classification and the EM algorithm. The evaluation is based on random samples of flows obtained from four large 24-hour trace files. We have investigated the performance of clustering using the following applications: FTP data, SSH, DNS, SMTP, Web, Napster, AOL messenger and Half-life. We have shown that some separation between the different applications can be achieved and some applications can be separated more effectively (e.g. Half-life) than others (e.g. web).

The authors of [20] use a similar approach based on the Naïve Bayes classifier and a large number of flow attributes. Although only one data set is used the flows have been hand-classified allowing a very accurate

evaluation. In their study it is not clear what influence the different attributes have on the classification.

Research on combining different non-ML techniques to identify network applications is presented in [21]. The work is based on the same hand-classified data set as [20]. The authors found that high classification accuracy can only be achieved using a combination of techniques. They also find that using the server port alone allowed for an average of 70% of the flows and bytes to be classified correctly. Some individual applications have higher accuracy for instance 80% for web traffic and 96% for mail traffic. However, high classification rates (>95%) could only be achieved using packet content in the classification process.

The authors of [22] have developed a method that characterises host behaviour on different levels to classify network traffic into different application types. The social level captures the behaviour of a host in terms of the number of other hosts it is communicating with. At the functional level the functional role of a host is considered, for example if a host is a provider or consumer of a service. At the application level the transport layer interactions between hosts is used. An evaluation based on three real traces shows that 80%-90% of the traffic could be identified with more than 95% accuracy.

V. Feature Selection Techniques

A feature set describing a data instance might range in size from two to several hundred features. The representative quality of this feature set greatly influences the effectiveness of the ML algorithm. It is therefore desirable to carefully select the number and type of features used to train the ML algorithm, a process known as feature selection. The benefits of feature selection are two-fold. Reducing the number of features decreases learning and classification times, while the removal of irrelevant or redundant features can also increase the classification accuracy.

Feature selection algorithms can be broadly classified into the filter model or wrapper model [23]. Filter model algorithms use a custom metric to rate and select features and provide feature subsets that are not biased towards any particular ML algorithm. The wrapper method evaluates the performance of different subsets using specific ML algorithms hence subsets are optimised towards the algorithm used. A number of subset search techniques can be used to generate feature subsets for the evaluators (see section V.B).

In this study we use the Correlation-based Feature Selection (CFS) filter technique. For further information regarding the use of other filter and wrapper methods for traffic flow statistics, see [24].

A. *Correlation-Based Feature Selection (CFS)*

The CFS algorithm [25] uses an evaluation heuristic that examines the usefulness of individual features along with the level of inter-correlation among the features. High scores are assigned to subsets containing attributes that are highly correlated with the class and have low inter-correlation with each other.

Conditional entropy is used to provide a measure of the correlation between each feature and the class and between features within the class. If $H(X)$ is the entropy of a feature X and $H(X|Y)$ the entropy of a feature X given the occurrence of feature Y the correlation between two features X and Y can then be calculated using the symmetrical uncertainty:

$$C(X \mid Y) = \frac{H(X) - H(X \mid Y)}{H(Y)} \quad (1)$$

The class of an instance is considered to be a feature. The goodness of a subset is then determined as:

$$G_{subset} = \frac{k\overline{r_{ci}}}{\sqrt{k + k(k-1)\overline{r_{ii}}}} \quad (2)$$

where k is the number of features in a subset, $\overline{r_{ci}}$ the mean feature correlation with the class and $\overline{r_{ii}}$ the mean feature correlation. The feature-class and feature-feature correlations are the symmetrical uncertainty coefficients (Equation 3).

B. *Search Techniques*

CFS as used in the study requires a search algorithm to provide subsets from the feature space. The following common search techniques were used:

- Greedy
- Best First
- Genetic

The Best First and Greedy search techniques require a starting point and search direction to be specified. We use forward and backward searches. A search that begins with zero features and increases in size on each iteration is known as a forward search. Starting with all features and reducing the subset size on following iterations is known as a backward search.

Greedy

Greedy search considers changes local to the current subset through the addition or removal of features. For a given 'parent' set, a greedy search examines all possible 'child' subsets through either the addition or removal of features. The child subset that shows the highest goodness measure then replaces the parent subset, and the process is repeated. The process terminates when no more improvement can be made.

Best First

Best First search is similar to greedy search in that it creates new subsets based on the addition or removal of features to the current subset. However, it has the ability to backtrack along the subset selection path to explore different possibilities when the current path no longer shows improvement. To prevent the search from backtracking through all possibilities in the feature space, a limit is placed on the number of non-improving subsets that are considered. In our evaluation we chose a limit of five.

Genetic

A Genetic search attempts to find an optimal solution using evolutionary concepts [26]. An initial population of individuals (solutions) is generated at random or heuristically. In every evolutionary step, known as a generation, the individuals in the current population are decoded and evaluated according to some predefined quality criterion (fitness function). To form a new population (the next generation), individuals are selected according to their fitness. Population selection schemes ensure that only high-fitness (good) individuals stand a better chance of 'reproducing', while unsuitable individuals are more likely to disappear.

Selection alone cannot introduce any new individuals into the population, i.e. it cannot find new points in the search space. These are generated by genetically inspired operators, of which the most well known are crossover and mutation. We chose an initial random subset population of 20, and performed 20 evolutionary steps. The crossover probability used was 0.6, while the probability of subset mutation was 0.033.

VI. MACHINE LEARNING ALGORITHMS

We use a number of supervised ML algorithms. As previously described, a supervised algorithm forms a model based on training data and uses this model to classify unseen new data. In this study we focus on well-known algorithms that have shown good results when applied to other problems. Furthermore, we focus on algorithms that do not require 'magic' parameters to be tuned. Therefore we do not test neural networks or support vector machines, as even though very good results might be obtained, optimisation of the many parameters is tedious and risks biasing the classifier to the training dataset. We use the following algorithms:

- C4.5 Decision Tree
- Naïve Bayes
- Naïve Bayes Tree
- Bayesian Networks

The algorithms are briefly described in the following sections. For more detailed descriptions the reader is refereed to the related work.

A. *C4.5 Decision Tree*

The C4.5 algorithm [27] creates a decision model based on a tree structure of nodes, branches and leaves. Nodes in the tree represent features, with branches representing possible values connecting features. A series of nodes and branches is terminated with a leaf, which represents 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, as other decision tree learners, 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 test outcomes to separate S are assigned to the branches.

C4.5 uses two main types of tests each involving only a single attribute A. In case of discrete attributes the test is A=? with one outcome for each value of A. For numeric attributes the test is $A \le \theta$ 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.

To find the optimal partitions of S C4.5 relies on greedy search and selects the candidate test set that maximizes a heuristic splitting criterion. C4.5 uses entropy based gain ratio to select the best split. If S_{Cj} is frequency of instances in S that belong to class C_j the information content that identifies the class of an instance in S is:

$$I(S) = -\sum_j S_{Cj} \log(S_{Cj}) \quad (3)$$

After S is partitioned into subsets by a test T the information gain is:

$$G(S,T) = I(S) - \sum_i \frac{|S_i|}{|S|} I(S_i) \quad (4)$$

A problem with this test is that it favours large numbers of partitions e.g. $G(S,T)$ is maximised by a test in which each S_i contains a single instance. The gain ratio criterion sidesteps this problem by taken the gain ratio of the partition into account:

$$\frac{G(S,T)}{P(S,T)} = \frac{G(S,T)}{-\sum_i \frac{|S_i|}{|S|} \log\left(\frac{|S_i|}{|S|}\right)} \quad (5)$$

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 close fitting of the training data leads to a decrease of the prediction

accuracy on unseen instances. C4.5 attempts to generalise the classifier by removing some structure from the tree after it has been built. Pruning is based on the 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 and therefore not useful. Instead of using the real error the C4.5 pruning algorithm uses a more conservative estimate. This estimate 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 prunes a tree in a single bottom up pass. For each tree C4.5 computes the upper bound of the error for the whole tree and the errors for all sub trees and leaves. Subtree replacement will replace the tree with a single leaf if the estimated error for the whole tree is lower than the weighted sum of the errors of the leaves. Subtree raising will lift a subtree if the estimated error of the new tree will be less than the estimated error of the subtree.

In our test the confidence level is 0.25 and the minimum number of instances per leaf is set to two. We use subtree replacement and subtree raising when pruning.

B. *Naïve Bayes*

Naïve Bayes is based on the Bayesian theorem [28]. This classification technique analyses the relationship between each attribute and the class for each instance to derive a conditional probability for the relationships between the attribute values and the class. We assume that X is a vector of instances where each instances is described by attributes $\{X_1,...,X_k\}$ and a random variable C denoting the class of an instance. Let x be a particular instance and c be a particular class.

Using Naïve Bayes for classification is a fairly simple process. During training, the probability of each class is computed by counting how many times it occurs in the training dataset. This is called the prior probability $P(C=c)$. In addition to the prior probability, the algorithm also computes the probability for the instance x given c. Under the assumption that the attributes are independent this probability becomes the product of the probabilities of each single attribute. Surprisingly Naïve Bayes has achieved good results in many cases even when this assumption is violated.

The probability that an instance x belongs to a class c can be computed by combining the prior probability and the probability from each attribute's density function using the Bayes formula:

$$P(C=c \mid X=x) = \frac{P(C=c)\prod_i P(X_i = x_i \mid C=c)}{P(X=x)} \tag{6}$$

The denominator is invariant across classes and only necessary as a normalising constant (scaling factor). It can be computed as the sum of all joint probabilities of the enumerator:

$$P(X=x) = \sum_j P(C_j)P(X=x \mid C_j) \tag{7}$$

Equation 8 is only applicable if the attributes X_i are qualitative (nominal). A qualitative attribute takes a small number of values. The probabilities can then be estimated from the frequencies of the instances in the training set. Quantitative attributes can have a large number (possibly infinite) of values and the probability cannot be estimated from the frequency distribution. This can be addressed by modelling attributes with a continuous probability distribution or by using discretisation. Discretisation transforms the quantitative attributes into qualitative attributes, and avoids the problem of using a continuous probability density function that does not match the true density. For this reason we evaluate Naïve Bayes using attribute discretisation.

C.*Bayesian Networks (Bayes Net)*

A Bayesian Network [29] is a combination of a directed acyclic graph (DAG) of nodes and links, and a set of conditional probability tables. Nodes can represent features or class, while links between nodes represent the relationship between them.

Conditional probability tables determine the strength of the links. There is one probability table for each node (feature) that defines the probability distribution for the node given its parent nodes. If a node has no parents the probability distribution is unconditional. If a node has one or more parents the probability distribution is a conditional distribution where the probability of each feature value depends on the values of the parents.

Learning in a Bayesian network is a two-stage process. First the network structure B_s is formed (structure learning) and then probability tables B_p are estimated (probability distribution estimation).

We use a local score metric approach that aims to optimise the network structure based on the quality of nodes as indicated by a given metric. The quality of the whole network is then the sum of the individual nodes. A local search algorithm is used to compute the metrics for each node.

The search algorithm we use is the K2 hill climbing algorithm created by G. Cooper and E. Herskovitz. This algorithm searches through the nodes according to some predetermined ordering of the data. The implementation we use searches in order of appearance of the features in the dataset. The K2 algorithm requires that the input data is discretised.

We use the Bayesian Metric in conjunction with the K2 search to determine node quality. The quality Q for a network structure Bs for a given database D is given by

$$Q(B_S, D) = P(B_s)\prod_{i=0}^{n}\prod_{j=1}^{q_i}\frac{(r_i - 1)!}{(r_{i-1} + N_{ij})!}\prod_{k=1}^{r_i} N_{ijk}! \quad (8)$$

where n is the number of features, r_i the cardinality (the number of elements in the set) of the ith feature x_i and q_i denotes the cardinality of the parent set of x_i in B_s, which is defined as the product of the cardinalities of all parent nodes of x_i. N_{ij} is the number of instances in D for which the parent set of feature i takes its jth value. N_{ijk} is the number of instances in D for which the parent set of feature takes its jth value and the feature i takes its kth value. The prior probability $P(B_s)$ of the network structure B_s is assumed to be constant and therefore ignored. See [29] and [30] for a detailed explanation of this algorithm.

An estimation algorithm is used to create the conditional probability tables for the Bayesian Network. We use a simple estimator, which estimates probabilities directly from the dataset. The simple estimator calculates class membership probabilities for each instance, as well as the conditional probability of each node given its parent node in the Bayes network structure.

Our combination of structure learning and search technique is only one of many combinations that can be used to create Bayesian Networks.

D. *Naïve Bayes Tree (NBTree)*

The NBTree [30] is a hybrid of a decision tree classifier and a Naïve Bayes classifier. Designed to allow accuracy to scale up with increasingly large training datasets, the NBTree algorithm has been found to have higher accuracy than C4.5 or Naïve Bayes on certain datasets. The NBTree model is best described as a decision tree of nodes and branches with Bayes classifiers on the leaf nodes.

As with other tree-based classifiers, NBTree spans out with branches and nodes. Given a node with a set of instances the algorithm evaluates the 'utility' of a split for each attribute. If the highest utility among all attributes is significantly better than the utility of the current node the instances will be divided based on that attribute. Threshold splits using entropy minimisation are used for continuous attributes while discrete attributes are split into all possible values. If there is no split that provides a significantly better utility a Naïve Bayes classifier will be created for the current node.

The utility of a node is computed by discretising the data and performing 5-fold cross validation (see Section III.B) to estimate the accuracy using Naïve Bayes. The utility of a split is the weighted sum of the utility of the nodes, where the weights are proportional to the number of instances in each node. A split is considered to be significant if the relative (not the absolute) error reduction is greater than 5% and there are at least 30 instances in the node.

VII. EXPERIMENTAL APPROACH

This section describes the data and features that form the basis of our study.

A. *Traffic Classes*

In traffic classification a class might be an individual application or a group of applications with similar characteristics. Example instances of each class are provided to the learning algorithm at training time. For accurate classification, the class instances used for training must be truly representative of the class. As our focus is on classifying game traffic, the classes are based on individual games.

The majority of game traffic classes were from the First Person Shooter (FPS) game genre, as these are most sensitive to QoS issues. A single real-time strategy (RTS) game was also included for comparison.

An additional 'other' class was also created, consisting of a sample of flow data taken from three publicly available trace files. This class was included to represent generic unknown network data other than online games, and includes web, email, DNS and peer-to-peer traffic, among others.

Table 1 summarises the classes used in the experiments.

Table 2: Traffic classes used in experiments

Class	Description	Genre/platform
CCG	Command and Conquer: Generals	RTS / PC
HL1	Half-Life: Death Match	FPS / PC
HL2-DM	Half-Life 2: Death Match	FPS / PC
HL2-CS	Half-Life 2: Counter Strike	FPS / PC
Q3	Quake 3 Death Match	FPS / PC
TS	Time Splitters	FPS / X-Box
HALO	Halo: Death Match	FPS / X-Box
HALO2	Halo 2: Death Match	FPS / X-Box
Other	Common network protocols e.g. HTTP, DNS, P2P	-

B. *Features*

Features are attributes that as a set describe an instance of a class. Here each instance represents a traffic flow generated by one of the classes. We use NetMate [32] to process packet traces, classify packets and compute features. 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.

Flows have limited duration. UDP flows are terminated by a 60 second flow timeout, while TCP flows are terminated upon proper connection teardown (TCP state machine) or after a 60 second timeout (whichever occurs first). 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 TCP connection attempts) or 'unsuccessful' flows (e.g. requests without responses). Flows devoid of payload cannot 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 distinguish active and idle periods of flows 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. The duration of idle periods is the time difference between a packet and the last packet at the beginning of the idle periods. The duration of active periods is the time difference between the last and first packet of the active periods. When a flow is terminated by timeout the time from the last packet until the timeout is not counted as idle time.

We compute the following features: protocol, duration, volume in bytes and packets, average volume in bytes and packets per active period, number of packets with push flag set (only for TCP – always 0 for UDP), packet length (minimum, mean, maximum, standard deviation), inter-arrival times (minimum, mean, maximum, standard deviation) active and idle times (minimum, mean, maximum, standard deviation). Aside from protocol and duration all features are computed separately in both directions of a flow. Packet length derived features are based on the IP length excluding link layer overhead. Inter-arrival times are computed with microsecond precision. All of the 36 features can be efficiently computed solely from the packets collected within each individual flow. Server port is not used as an attribute as 'wrong' ports could introduce an unknown bias.

C. *Data Traces*

Where possible game data was obtained from public game servers. This is the most realistic and diverse data as it is based on a large number of real players. For games where it is impossible to run a public server (Xbox, CCG) and Q3 (which is no longer widely played), we used traces captured in controlled experiments designed to obtain game traffic characteristics.

Due to the client/server discovery mechanisms used by first person shooter games, additional care is needed when sampling data from public servers to ensure actual game traffic is obtained. As described in [33], much of the traffic seen by public servers is from game clients probing the server for information, such as round trip times, current map and number of current players. We call this traffic probe traffic as opposed to non-probe traffic, which is traffic that is exchanged when players are actually playing

Half-Life 1 traffic was captured over 15 games at a server located in the Swinburne University LAN [34]. Between 3 and 7 players were present in each game. Some HL1 traffic was also captured on CAIAs public HL1 counter-strike server (now offline). As the majority of these flows were probe flows, we have taken a random sample equal in size to the number of non-probe flows.

Half-Life 2 DM and CS traffic was captured over a period of one month from CAIAs public Half-Life 2 server [35]. Due to a large amount of probe flows these datasets were constructed using stratified sampling on the flow size. We took all flows where more than 10 packets were sent from client to server and an equal amount of short flows randomly sampled from the population of a few million probe flows. This strategy ensures that the training set is not overly large and has a balance of small and large flows for each game.

Quake 3 game traffic was captured on a server located on the university LAN [36]. Games captured consisted of between 2 and 8 players.

Halo traffic was captured between up to three Xbox consoles connected via System Link, for games consisting of up to four players and lasting up to 50 minutes [38]. Halo 2 traffic was captured from between up to four Xboxes connected to a Hub and an Xbox server [39]. Time Splitters traffic was captured between three Xbox consoles over six games with up to nine players participating [40].

Xbox units communicating over System Link are differentiated by Ethernet MAC address (the UDP/IP payloads within each Ethernet frame contain a non-functional source and destination IP set to 0.0.0.1). Xbox game servers often generate broadcast packets during the initial stage of creating a client and server connection. These were removed from the Xbox datasets.

The 'Command and Conquer: Generals' data was obtained from Mark Claypool's archive of game traffic [9]. The traces consist of a number of 15-45 minute games.

The data used to represent the 'other' class was drawn from publicly available NLANR [37] data traces. These traces were captured in different years and at different locations. We used flow data from four 24-hour

periods of these traces (auckland-vi-20010611, leipzig-ii-20030221, nzix-ii-20000706).

As a 24-hour period of the packet traces contains up to several million flows, our data set is sampled from the total number of flows. We randomly sampled 200,000 flows from each trace, for a total of 600,000 flows. As the public traces do not contain any payload information we use port numbers to estimate the different applications contained within 'other' class (see Table 3). 75-80% of the traffic flows and 70-75% of the volume in the 'other' class can be attributed to these few popular applications.

Table 3: Several common applications in 'other' class

Application	Description	Protocol	Flows (%)	Volume (%)
HTTP	World Wide Web	TCP	52.45	52.37
DNS	Domain Name Service	UDP/TCP	12.73	1.01
eDonkey	P2P File sharing	UDP/TCP	10.28	8.24
HTTPS	Secure Web	TCP	3.34	2.43
SMTP	E-Mail	TCP	1.92	6.04
Kazaa	P2P file sharing	TCP/UDP	1.08	4.72
POP	Electronic Mail	TCP	1.04	0.51

As the majority of online games (and all of those tested here) use UDP/IP, it was important that the non-game class contained a sizeable portion of UDP traffic. There were approximately 126,568 UDP flows in the 'other' class, accounting for 21% of the total (and far exceeding the total game flows). We removed all flows with ports that were the default Half-life 1/2, Quake 3 or Xbox ports from these traces. A summary of all the trace data is shown in Table 4.

Table 4: Trace summary

Class	Flows	Volume (MB)	Protocol
CCG	561	252	UDP
HL1	1,902	592	UDP
HL2-DM	10,519	25,570	UDP
HL2-CS	2,391	1,641	UDP
Q3	1,246	83	UDP
TS	23	23	UDP
HALO	35	272	UDP
HALO2	78	126	UDP
Other	600,000	6,993	TCP/UDP

We arranged the trace data into three different datasets, each containing game data and non-game (other) data. The first contains each individual game as separate class (game), the second contains one class per game engine (engine), while in the final trace all games are aggregated into a single class (2-class). Note that the engine dataset is essentially identical to the game dataset, with the HL2-DM and HL2-CS combined into one class.

D. *ML Software*

Experiments were conducted using the WEKA (Waikato Environment for Knowledge Analysis) software version 3.4.4 [41]. Widely used in the academic community, WEKA contains Java implementations of all the algorithms described above.

VIII. RESULTS AND ANALYSIS

First we evaluate accuracy, precision and recall of the different algorithms. Then we investigate the influence of feature selection, using CFS and different search techniques. We also present what features are more useful according to our feature selection technique used. Finally, we evaluate the performance of the different algorithms in terms of classification and training speed.

A. *Algorithm Evaluation*

Figure 2 shows the accuracy of each algorithm and dataset using the entire feature set. Overall accuracy is greater than 99% for each of the algorithms when using all features, except for Naïve Bayes. The 2-class dataset provided the highest accuracy for each of the algorithms. The per-engine dataset provided slightly higher accuracy than the per-game dataset.

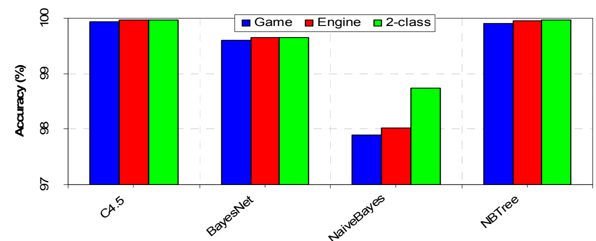

Figure 2: Accuracy using all features

Overall accuracy is quite good starting reference, but it does not give us an indication as to the accuracy of each individual class. It can also be biased towards classes with a greater number of instances. For example, consider a dataset containing only HL2-DM (10,519 instances) and TS (23 instances). So long as the recall of the HL2-DM class is high, overall accuracy will also be high, regardless of whether recall for TS was high or low.

Therefore, in the following we evaluate the recall and precision of all individual classes and use mean recall and precision values instead of accuracy. Recall and precision are also calculated according to the total volume of each class correctly or incorrectly classified.

It is important to consider these metrics according to volume as during training algorithms optimise the classifiers based on the number of instances in the dataset. As game traffic is often a combination of many probe flows (with small volume) and fewer in-game flows (with larger volume), it is very important to also consider precision and recall calculated according to bytes classified. These rates are calculated by adding the volume of each instance into a standard confusion table. By comparing the instance-based metrics and volume-based metrics we can determine the proportion of actual in-game flows that are correctly classified.

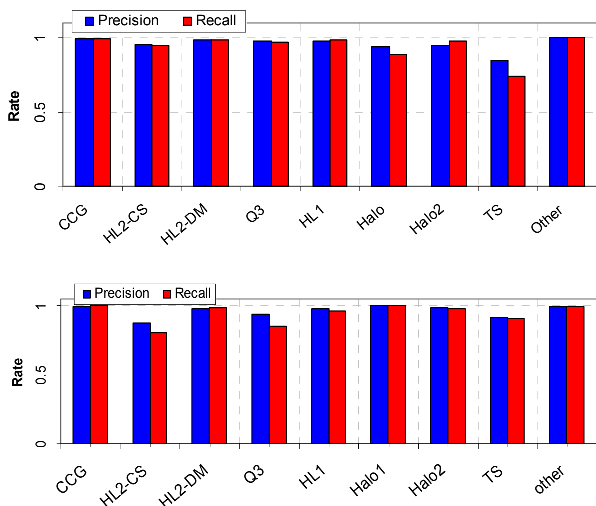

Figure 3: Precision and recall by instance (top) and volume (bottom) using C4.5 with all features, per-game dataset

Figure 3 plots the precision and recall rates by for each traffic class using the per-game dataset and C4.5 algorithm. The majority of classes have high instance-based precision and recall, which indicates low numbers of false positives and negatives. Classes with very few training instances, such as Time Splitters and Halo, still achieve acceptable recall and precision rates (0.73 and 0.89 respectively). Considering volume, there is an immediate difference seen in HL2-DM and HL2-CS, Q3 and HL1. The reduced precision and recall for the HL2-based games is due to a number of HL2-CS in-game flows being misclassified as HL2-DM. A number of Q3 flows with larger volumes were also classified as HL2-DM. The remaining classes are classified at rates equal or better than those based on flow instances.

Figure 4 plots the precision and recall rates for each traffic class using the per-game dataset and Bayes Net algorithm. The instance-based precision rates for the FPS games are significantly lower than those using C4.5, indicating an increase in the number of false positives and false negatives between these classes. The RTS and 'other' class are both separated with high precision and recall. Volume-based metrics are again quite different, with fewer game flows from HL2-DM being classified as HL2-CS (as indicated by the high precision of HL2-CS). However many HL2-CS game flows are being classified as HL2-DM. Besides Counter Strike, many of the applications were classified with high precision and recall in terms of volume.

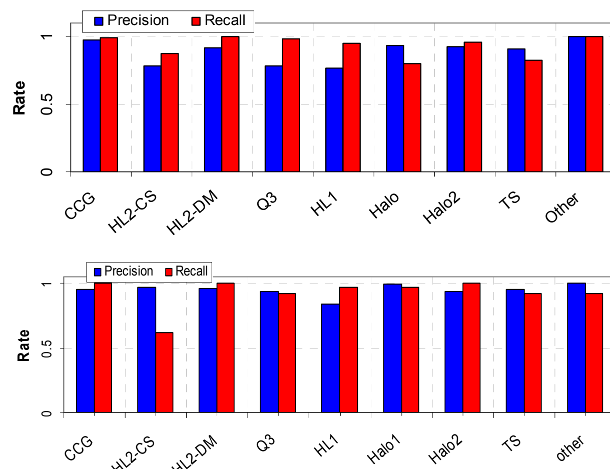

Figure 4: Precision and recall by instance (top) and volume (bottom) using Bayes Net with all features, per-game dataset

Figure 5 plots the precision and recall rates with all features for each traffic class using the per-game dataset and Naïve Bayes algorithm. The sample size of a class appears to have a large affect on the instance-based precision and recall rates when using Naïve Bayes. Classes with few training instances perform very poorly, with Time Splitters approaching zero for precision and recall. Again Command and Conquer Generals appears distinct enough from the FPS games to be classified with high accuracy, as is the case with the generic 'other' class.

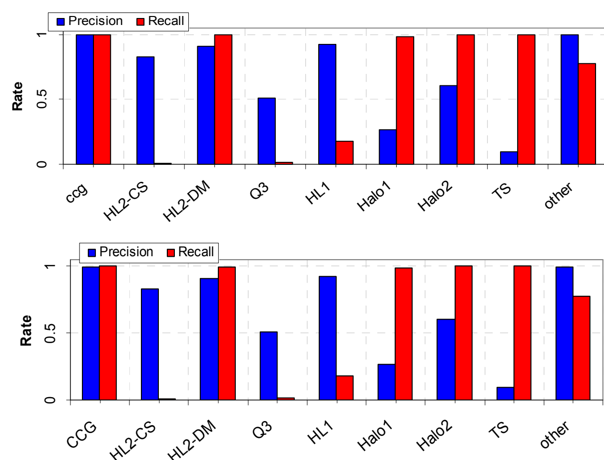

Figure 5: Precision and recall per class using Naïve Bayes with all features, per-game dataset

As with the flow-based metrics, the volume-based rates for the FPS game classes are quite varied. Much of Q3 was classified as Halo1, while most HL2-CS game flows are classified as HL2-DM and HL1. Flows from the 'other' class were misclassified as all of the game

classes, reducing the precision of each (with the exception of CCG).

Figure 6 plots the precision and recall rates for each traffic class using the per-game dataset and NBTree algorithm. Instance-based performance for NBTree is on par with C4.5 and in some cases slightly better. Interestingly a number of Halo flows were classified as Command and Conquer flows, reducing the CCG precision and the recall of the Halo class. This crossover between RTS and FPS classes was not seen in tests with the other algorithms.

The volume-based results were also comparable to those seen in C4.5, although will slightly less recall and precision for most classes.

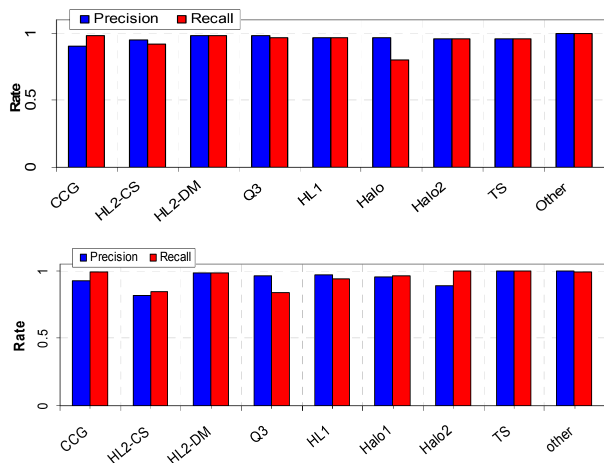

Figure 6: Precision and recall per class using NBTree with all features, per-game dataset

Figure 7 summarises the mean recall and precision rates of the algorithms when using the per-game dataset. Rates for C4.5 and NBTree show almost identical performance (~0.95), while Bayes Net (~0.91) was slightly less accurate.

The precision of Naïve Bayes is very poor, and it appears this algorithm has difficulty with disparate class sizes. For C4.5, Bayes Net and NBTree there does not appear to be a great difference when comparing flow instances and volume.

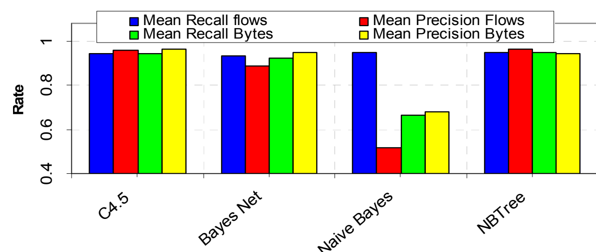

Figure 7: Mean recall and precision for each algorithm, flows and bytes, game dataset

The only difference between the per-engine and per-game datasets was the aggregation of the two HL2-based games. Therefore in Figure 8 we compare the instance-based recall and precision for the single HL2 class against the mean rates of the individual HL2-CS and HL2-DM classes. As a single class the recall and precision rate improves slightly. This test shows that two separate game modifications with distinct game mechanics but the same underlying engine can be detected not only individually but also as a single class. Using a single class appears to lead to higher accuracy.

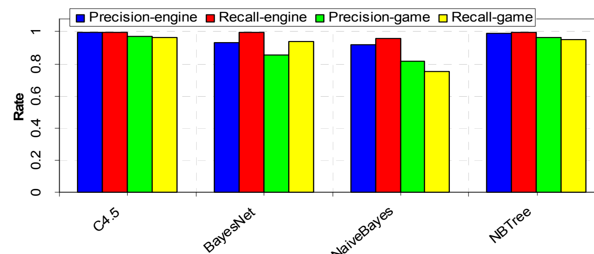

Figure 8: Half-Life 2 precision and recall with all features

Figure 9 compares the mean recall and precision of the algorithms when using the 2-class dataset. Precision and recall metrics are generally higher than for the per-game dataset.

Apart from Naïve Bayes, there is very little separating the different algorithms, as all are very accurate for flows and volume. The recall and precision metrics for flows and bytes approached .99 for C4.5 and NBTree. At 0.95 the Bayes Net volume-based recall stood out, being .04 (4%) less than for flow-based recall.

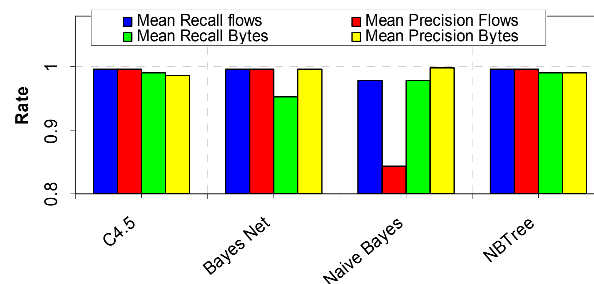

Figure 9: Mean recall and precision for each algorithm, flows and bytes, 2-class dataset

Even though overall accuracy was slightly optimised towards the class with the largest number of instances (other), the difference in recall for the two classes was minor.

B. *Feature Selection*

Feature subset selection was performed on all three datasets using CFS with Best First, Genetic and Greedy search methods (see section V). Figure 10 shows the average size of the selected subsets as a percentage of the full feature set, averaged across the datasets.

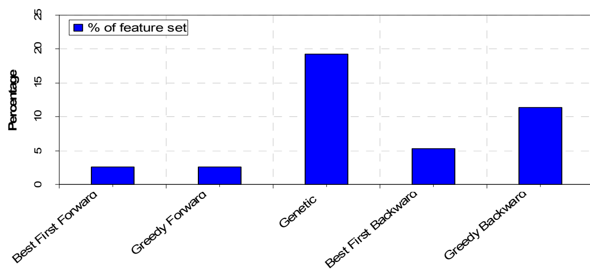

Figure 10: Percentage of full feature set by search method

CFS provides substantial reductions in the number of features used, with the largest subset using 20% of the features. The forward directional searches were very aggressive, with both Best First and Greedy Forward selecting only one feature for each dataset (minimum forward packet length).

Figure 11 shows the change in overall accuracy averaged across the different search methods when using CFS for each of the algorithms and datasets, compared to accuracy using all features.

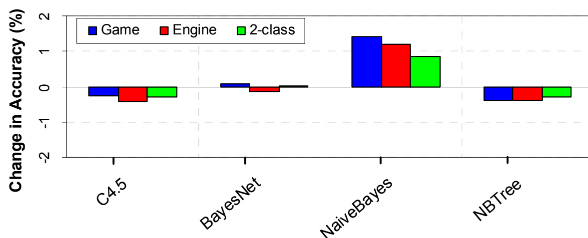

Figure 11: Average change in overall accuracy by subset selection method compared against using full feature set

With the exception of Naïve Bayes it appears that CFS selection causes a slight reduction in accuracy. Although classification improves for Naïve Bayes when using CFS, this is somewhat misleading, as overall accuracy is heavily biased towards classes with large numbers of instances. Again a more practical evaluation of CFS performance can be made by considering the per-class accuracy (i.e. recall).

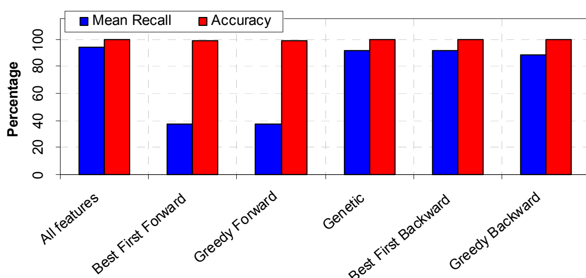

Figure 12: Mean recall compared to overall accuracy, per-game dataset

Figure 12 compares the mean class recall against accuracy, averaged across the algorithms for the per-game dataset. Recall is expressed as a percentage for comparison. The large difference between overall accuracy and mean class recall can be seen for several of the tests (such as greedy forward, best first forward).

Figure 13 shows the change in mean class recall for each algorithm when compared to the mean class recall obtained using the full feature set, for the per-game dataset. We see that the 60% drop in class recall ranges across all the algorithms when using the forward-directional searches. The poor mean class recall was due to classes with very few training instances such as 'Time Splitters' and 'Halo' having a recall of 0 as a consequence of using these search methods. A similar result was observed for the per-engine dataset.

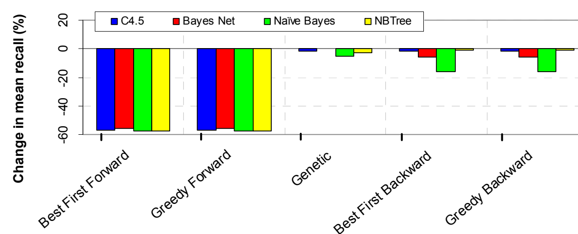

Figure 13: Change in mean class recall by CFS search method, 'game' dataset

Figure 14 compares the mean class recall against accuracy, averaged across the algorithms, for the 2-class dataset. The difference between overall accuracy and recall is less pronounced for the 2-class dataset. As both classes contain a significant number of flows, it appears difficult for the evaluator to optimise for one class without reducing the overall accuracy.

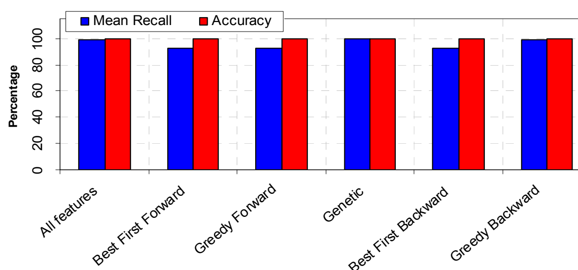

Figure 14: Mean recall compared to accuracy for 2-class dataset

Figure 15 again shows the change in mean class recall for each algorithm, this time using the '2-class' dataset. In most cases CFS selection decreases mean class recall as compared to using the full feature set, although not by greater than 10%. As suggested in Figure 14, using CFS subset selection in a 2-class scenario does not incur as severe a penalty than with a multi-game scenario (per-game, per-engine).

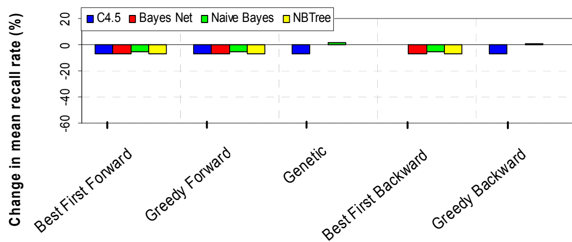

Figure 15: Change in mean class recall by CFS search method, '2-class' dataset

Overall the results of the CFS evaluation suggest that a reduced number of features still allow game traffic to be separated from generic IP network traffic.

There were several cases of extreme reductions in the feature space, though results suited only few classes with the highest number of training instances. A range of classes with an equal number of training instances would most likely require more features, but a significant reduction in features would still be expected.

The genetic search appeared to have the least impact of the average class accuracy, while still providing a large reduction in features and classification time. For the 2-class dataset, genetic search and CFS had a minimal impact, and improved the mean class recall when using Naïve Bayes.

C. *Features of Interest*

We also examine the frequency at which particular features are included in the selected feature subsets. This provides an excellent indicator as to which features are likely to be better at discriminating the classes. Figure 16 graphs the percentage of selected feature sets in which a given feature was included, for each of the datasets.

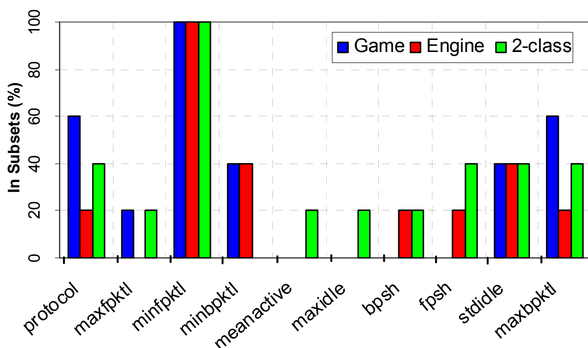

Figure 16: Percentage of subsets in which feature was selected

'Minimum forward packet length' (minfpktl) is clearly the strongest feature, appearing in all selected subsets. There does not seem to be any other feature that is as dominant for all datasets, although 'protocol', 'standard deviation idle time' (stdidle) and 'Minimum backward packet length' (minbpktl) are often selected.

There was some level of consistency in the subset of features selected by the different search methods across the datasets, indicating that the stronger features are relatively independent of dataset. The 2-class dataset tends to require more features than the other datasets (as neither class is particularly homogenous).

The results shown above are for CFS across all different search techniques. Different feature selection metrics may produce different results.

D. *Performance Evaluation*

To further differentiate the algorithms, we calculate two additional performance metrics: training and classification time (defined in Section III.B). These are evaluated using the game and 2-class datasets.

Performance metrics were measured using a 3.4 GHz Pentium 4 workstation with 4GB of RAM and the Java VM 1.5.0_04-b05.

Figure 17 shows the classifications per second for each of the algorithms for the per-game and 2-class datasets.

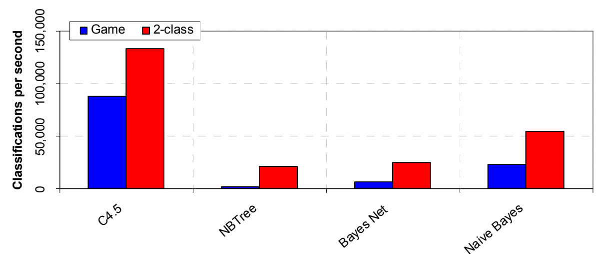

Figure 17: Classifications per second for 2-class and game datasets

In terms of classifications per second, C4.5 (88,233/133,786) has a considerable advantage in both datasets, and is markedly faster than the nearest algorithm, Naïve Bayes (23,238/54,580). NBTree (2,075/21,150) is the slowest algorithm.

Having only two classes provides a substantial increase in classification speed for the algorithms. C4.5 remained significantly faster than the others, although the gap between NBTree (21,150) and Bayes Net (25,020) narrowed.

Figure 18 shows the build time for each of the algorithms with the game and 2-class datasets. Note that the y-axis is non-linear to accommodate the long NBTree training period.

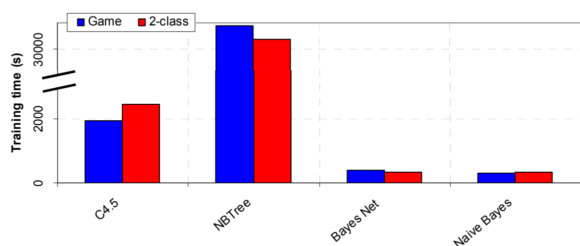

Figure 18: Training time for 2-class and game datasets

Naïve Bayes provided the fastest build time for the per-game dataset (302.14s), followed closely by Bayes Net (395.91s). C4.5 is significantly slower than Bayes Net, while NBTree (35,418.2s) is by far the slowest learner.

C4.5 and Naïve Bayes see an increase in training time for 2 classes, but are still the slowest learners. Bayes Net and NBTree take approximately the same time to create the classification model.

As CFS selection did not significantly affect classification accuracy and average recall for the 2-class dataset, we tested for any reduction in processing cost. Figure 19 shows the classification and training times for subsets as a percentage of the classification and training times when using the full feature space, for the 2-class dataset.

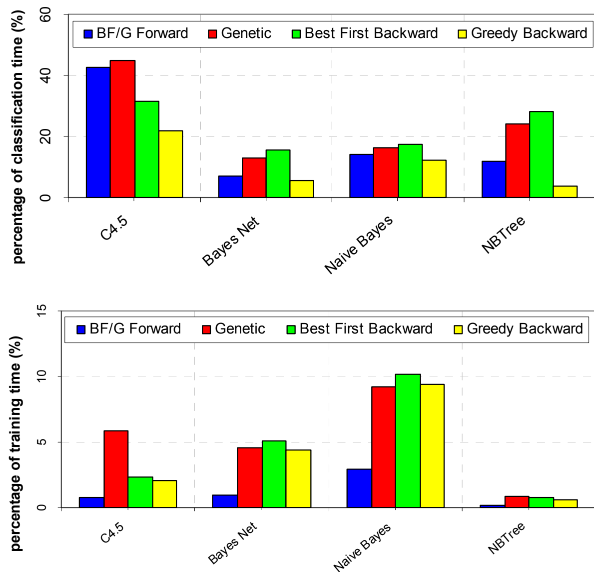

Figure 19: CFS subset classification (top) and training (bottom) times as a percentage of classification and training times when using all features

Each algorithm experiences a great improvement in classification speed, taking only 10-40% of the original time. Reductions are greater still for training times, at 10% the original training time and less. Although not shown here, the 'per-game' and 'per-engine' datasets saw a similar reduction in training and classification times.

E. *Limitations of the Datasets*

Imbalanced datasets generally pose a problem for machine learning algorithms. An imbalanced dataset contains training classes with a significant difference in the number of instances. This is problematic as algorithms optimise towards overall accuracy and thus 'favour' the large classes. However, it is difficult to obtain a large game traffic dataset especially for games where no public servers can be set up (Xbox games).

Balancing the dataset by reducing the dataset size would likely lead to an overestimation of accuracy due to a lack of variance in the feature distributions. Despite the imbalanced dataset used in the experiments, with many more non-game than game flows, we were able to detect game flows with high accuracies (even for classes with few instances) with three of the four algorithms evaluated.

The results obtained thus far are very promising, but there are a number of avenues to be further explored. The data traces for the games were either taken from a single public server or from some controlled lab experiments. Although we were able to obtain data for a large number of players from the public server it is likely that at least the probe flows introduce some bias in our study that lead to an increased accuracy. If we were to capture probe traffic on a large number of servers there would likely be more variation in the feature value space as some features (for example packet lengths) are indirectly dependent on game server configuration.

For example, as the server's response contains information such as sever name, packet length of the server to client traffic would have a wider distribution if more servers (with different names) were considered. A more complete approach may be to combine data from several games servers and possibly clients. An interesting observation however is that the more popular features were either in the forward direction (client to server) or were independent of direction (protocol, active and idle times). This may have reduced the influence of using data from a single server, although the more complete approach is still preferred.

For several of the controlled experiments there may also be bias due to factors such as a lack of variation of player combinations and maps played.

IX. Conclusions and Future Work

In this paper we have evaluated different machine learning algorithms for the purpose of classifying online games based on statistical payload-independent flow attributes. We have shown that games can be separated both from each other and from common network traffic. With a feature space of 36 flow statistics we were able to achieve overall classification accuracies of greater than 99%, and individual class accuracies greater than 95%.

In testing with the CFS evaluator, packet length statistics were identified as the strongest class discriminators. We also identified a weakness in using feature selection when classes have a disparate number of training instances. As subsets are evaluated according to overall accuracy, they are optimised to provide high accuracy for classes with the most instances. An equal amount of training instances for each class would likely produce higher per-class accuracy, although with a slightly larger feature set. The reduced feature sets tested provided significant reductions in classification and training time, and the ability to trade off accuracy for speed is a possible option.

Algorithm performance was quite consistent across the datasets. Whether games were grouped into individual classes, by engine or as individual classes, classification accuracy remained high across all the algorithms. Byte-based results were also good, though flow-based results were generally better. We believe the byte accuracy could be further improved as algorithms optimise classifiers only on the number instances during training. Increased per-byte accuracy may be obtainable if a classifier is trained with flows given different weights based on volume.

Particularly accurate algorithms were C4.5 and NBTree, which were able to provide high recall and precision for classes, even those with low numbers of training instances. The classification speed and training times for NBTree do not appear suitable for real-time operation however. Bayes Net, while not as accurate as either C4.5 or NBTree, offers very good accuracy and fast build times. Although classification speed is not exceptional, it has shown to increase in performance when using smaller subsets of features.

Whether trying to classify games as individual applications or as a single category against non-game traffic, decision tree classifiers appear best suited to the flows statistics that we use. In particular the C4.5 algorithm provided good results across all of our tests and is also quite fast.

We plan to extend this study using other popular interactive applications, such as Internet telephony and streaming video, and identify the features capable of identifying these applications.

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] G. Armitage, "An Experimental Estimation of Latency Sensitivity in Multiplayer Quake3", Proceedings 11th IEEE International Conference on Networks (ICON) 2003, Sydney, Australia, September 2003.

[2] S. Zander, G. Armitage, "Empirically Measuring the QoS Sensitivity of Interactive Online Game Players", Australian Telecommunications Networks & Applications Conference (ATNAC) 2004, Sydney, Australia, December 2004.

[3] L. Stewart, G. Armitage, P. Branch, S. Zander, "An Architecture for Automated Network Control of QoS over Consumer Broadband Links", IEEE TENCON 05 Melbourne, Australia, 21 - 24 November 2005.

[4] Thomas Karagiannis, Andre Broido, Nevil Brownlee, kc claffy, "Is P2P dying or just hiding?" In Proceedings of Globecom 2004, November/December 2004.

[5] Tom M. Mitchell, "Machine Learning", McGraw-Hill Education (ISE Editions), December 1997.

[6] S. Zander, T.T.T. Nguyen, G. Armitage, " Automated Traffic Classification and Application Identification using Machine Learning", IEEE 30th Conference on Local Computer Networks (LCN 2005), Sydney, Australia, 15-17 November 2005.

[7] 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, Sicily, Italy, 2004.

[8] SONG – Simulating Online Network Games Database http://caia.swin.edu.au/sitcrc/song/index.html (January 2006)

[9] Mark Claypool's Homepage http://web.cs.wpi.edu/~claypool/ (December 2005)

[10] Instat: Online gaming anything but fun and games, http://www.instat.com/press.asp?ID=576&sku=IN030683A (January 2006)

[11] IANA Port Numbers, http://www.iana.org/assignments/port-numbers (January 2006)

[12] Ports database, http://www.portsdb.org/ (as of January 2006)

[13] Cisco IOS Documentation, "Network-Based Application Recognition and Distributed Network-Based Application Recognition",http://www.cisco.com/univercd/cc/td/doc/product/software/ios122/122newft/122t/122t8/dtnbarad.htm (as of January 2006).

[14] J. Frank, "Machine Learning and Intrusion Detection: Current and Future Directions", Proceedings of the National 17th Computer Security Conference, 1994.

[15] 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.

[16] 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, Sicily, Italy, 2004.

[17] 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.

[18] A. Soule, K. Salamatian, N. Taft, R. Emilion, and K. Papagiannaki, "Flow Classification by Histograms or How to Go on Safari in the Internet", In ACM Sigmetrics, New York, U.S.A., June, 2004.

[19] P. Cheeseman, J. Stutz, “Bayesian Classification (Autoclass): Theory and Results”, Advances in Knowledge Discovery and Data Mining, AAAI/MIT Press, USA, 1996.

[20] A. W. Moore and D. Zuev, “Internet Traffic Classification Using Bayesian Analysis Techniques”, ACM SIGMETRICS, Banff, Canada, June 2005.

[21] A. Moore, and K. Papagiannaki, “Toward the Accurate Identification of Network Applications” Passive & Active Measurement Workshop, Boston, U.S.A., April, 2005.

[22] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, “BLINC: Multilevel Traffic Classification in the Dark”, ACM Sigcomm, Philadelphia, PA, August 2005.

[23] M. A. Hall, G. Holmes, “Benchmarking attribute selection techniques for discrete class data mining“, IEEE Transactions on Knowledge & Data Engineering, Vol 15, No 6, pp 1437-1447, 2003.

[24] N. Williams, S. Zander, G. Armitage, "Evaluating Machine Learning Algorithms for Automated Network Application Identification", CAIA Technical Report 060410B, April 2006

[25] M. Hall, “Correlation-based Feature Selection for Machine Learning“, PhD diss. Department of Computer Science, Waikato University, Hamilton, NZ, 1998.

[26] D. E. Goldberg, "Genetic Algorithms in Search, Optimization, and Machine Learning", Addison-Wesley, 1989.

[27] R. Kohavi and J. R. Quinlan, Will Klosgen and Jan M. Zytkow, editors, “Decision-tree discovery”, In Handbook of Data Mining and Knowledge Discovery, chapter 16.1.3, pages 267-276, Oxford University Press, 2002.

[28] G. H. John, P. Langley, “Estimating Continuous Distributions in Bayesian Classifiers”, 11th Conference on Uncertainty in Artificial Intelligence, pp. 338-345, Morgan Kaufman, San Mateo, 1995.

[29] R. Bouckaert, “Bayesian Network Classifiers in Weka”, Technical Report, Department of Computer Science, Waikato University, Hamilton, NZ 2005.

[30] D. Chickering, D. Geiger, D Heckerman, „“Learning Bayesian Networks is NP-Hard” Technical Report MSR-TR-94-17, Microsoft Research, 1994.

[31] Ron Kohavi, “Scaling Up the Accuracy of Naïve-Bayes Classifiers: a Decision-Tree Hybrid”, 2nd International Conference on Knowledge Discovery and Data Mining (KDD), 1996.

[32] NetMate, http://sourceforge.net/projects/netmate-meter/ (October 2005).

[33] S. Zander, D. Kennedy, G. Armitage "Dissecting Server-Discovery Traffic Patterns Generated By Multiplayer First Person Shooter Games”, ACM NetGames 2005, Hawthorne NY, USA, 10-11 October, 2005

[34] T.Lang, G.Armitage, P.Branch, H-Y.Choo. "A Synthetic Traffic Model for Half Life**,**" Australian Telecommunications Networks & Applications Conference 2003, (ATNAC 2003), Melbourne, Australia, December 2003.

[35] CAIA Game Research: http://caia.swin.edu.au/genius/ genius-what.html (January 2006)

[36] T.Lang, P.Branch, G.Armitage. "A Synthetic Traffic Model for Quake 3," ACM SIGCHI ACE2004 conference, Singapore, June 2004

[37] NLANR traces: http://pma.nlanr.net/Special/ (Oct. 2005).

[38] M. D. Pozzobon, "Capturing Xbox System Link Traffic while playing Halo**,**" (html) CAIA Technical Report 020802A, August 2002

[39] J. Bussiere, S. Zander, "Empirical Measurements of Player QoS Sensitivity for the Xbox Game Halo2," CAIA Technical Report 050527A, May 2005

[40] A.M. Pavlicic, "Capturing Xbox System Link Traffic While Playing TimeSplitters2," CAIA Technical Report 031009A, October 2003

[41] WEKA 3.4.4, http://www.cs.waikato.ac.nz/ml/weka/ (October 2005).

[42] G. Cawley, “Efficient Sequential Minimal Optimisation of Support Vector Classifiers”, Technical Report, School of Information Systems, University of East Angila, Norwich, Norfolk, UK, 2001

[43] Put to the test: http://www.networkworld.com/news/2002/0415idsevad.html?net (January 2006)