ANGEL - Machine Learning Classification

Sebastian Zander
Centre for Advanced Internet Architectures, Technical Report 060301A
Swinburne University of Technology
Melbourne, Australia
szander@swin.edu.au

***Abstract*—The Automated Network Games Enhancement Layer (ANGEL) project aims to leverage machine learning techniques to automate the classification and isolation of interactive (e.g. games, voice over IP) and non-interactive (e.g. web) traffic. This information is then used to dynamically reconfigure the network to improve the Quality of Service provided to the current interactive traffic flows and subsequently deliver improved performance to the end users.**

In this document we first describe the current solutions for network traffic classification and their shortfalls. Then we present the novel approach of classifying network traffic flows based on machine learning techniques and statistical flow attributes such as packet length and inter-arrival time statistics. We evaluate if a machine learning based approach meets the accuracy and performance requirements of the ANGEL architecture and select appropriate machine learning algorithms. The evaluation is based on results obtained for real game traffic captured in the network.

I. Introduction

Highly interactive applications, such as First Person Shooter (FPS) games or Voice over IP (VoIP), have a narrow tolerance to delay, jitter and packet loss (see [1], [2]) necessitating more rigid Quality of Service (QoS) compared to the best effort service used for traditional Internet applications such as web or email. In order for QoS to be effective, an accurate and timely method of identifying and classifying interactive flows is required. As it is unlikely that end-host applications will ever explicitly signal their QoS demands to the network, the network itself must identify interactive flows and establish adequate QoS for these flows by giving them priority over other traffic in the network. The ANGEL system is an architecture capable of improving QoS for interactive applications. An early version of this architecture was presented in [3].

From February 2007 and July 2010 this report was a confidential deliverable to the Smart Internet Technologies CRC. With permission it has now been released to the wider community.

A key component of the ANGEL architecture is the flow classifier. Its purpose is classifying network flows into interactive flows and non-interactive flows. Current popular methods of identifying the network applications responsible for traffic flows are TCP/UDP port-based and packet 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-layer protocol while signature-based methods search for application specific byte sequences in the packet payload.

Port-based classification systems are moderately accurate at best and will become less effective in the near future. For example, a host (one IP address) running multiple servers of the same game must run all but the first on arbitrary port numbers rather than the specified default port, making server port-based classification unreliable. Many game clients are behind Network Address Translators (NATs), which may change the client port. We have studied the distribution of client and server ports for a specific FPS game and found that a significant number of flows could not be identified purely based on default port numbers [4].

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. Protocol decoding for games would require a significant amount of reverse engineering as the game protocols are not openly specified to prevent users from cheating.

Machine learning (ML) techniques [5] provide a promising alternative through classifying flows based on application protocol (payload) independent features such as packet length and inter-arrival time statistics. 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].

We have previously used a wide range of machine learning algorithms to separate common network applications, such as web and mail, from traffic produced by different games (see [7] and [8]). The requirements for the ANGEL flow classifier have been specified in the ANGEL architecture [9]. In this paper we evaluate if machine learning based classification algorithms would be suitable for ANGEL. Our evaluation is based on real game traffic captured in the network for a number of popular FPS games.

We find that some ML algorithms are able to separate the different games from each other and other traffic with very high ($\geq$99%) accuracy. Results for separating game fron non-game traffic were good ($\geq$90%) even when using a reduced number of features after feature selection. Preliminary results suggest that game flows can be classified with high accuracy within 1-2 seconds of their lifetime, which is fast enough for typical game applications [10].

We also find that some of the accurate ML techniques are 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 are able to train fast enough allowing for frequent updates of the classifier (training times of less than half an hour).

The paper is structured as follows. Section 2 discusses the shortfalls of current classification approaches and outlines our machine learning approach. Section 3 evaluates if ML algorithms can be used in the ANGEL system to identify interactive game traffic. Section 4 concludes and outlines future work.

II. ML-BASED CLASSIFICATION

First we describe the currently used techniques and explain their shortfalls. Then we describe the ML-based approach.

A. Current Classification Techniques

1) 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 become larger since the port number range is not likely to increase.

Even applications with well-known or registered ports can end up using different port numbers when users attempt to hide their application's existence or bypass port-based filters, or when multiple servers are sharing a single IP address (host). Furthermore some applications (e.g. passive FTP or video/voice communication) use dynamic ports unknowable in advance.

Online games usually only have a commonly known default port (but no IANA registered port), which means other non-game applications are potentially using the same port number. Often online gaming servers run multiple server applications on a single physical host (IP address), which means all servers but the first must run on different ports. Therefore many servers run on non-default ports. Usually game clients use the default port but many are behind NATs that can change the port number. A study in [4] shows that for a particular FPS game a significant number of flows could not be identified based on the default port number.

2) 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. 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 very reliable classification are high.

With game traffic this approach is also difficult to realise, as there are many existing games and their protocols are usually not openly specified (to prevent players from cheating) and high effort would need to be spend into reverse engineering.

3) Signature-based Approaches: To overcome the inefficiencies of protocol decoding some researchers have proposed to use signature-based methods. These methods search for specific application-characteristic patterns in the payload of the packets. The advantage of this approach is that it is more effective than pure port-based classifications and more efficient than protocol decoding.

On the other hand signature-based methods are less accurate than protocol decoding. However, these methods are still protocol dependent as signatures are application-specific and must be developed with a protocol specification or through reverse engineering. In the past developing signatures for some unspecified binary protocols has been found to be difficult. Overall, signature-based methods provide a very good trade-off between resource efficiency and classification performance, but there are a number of ways to defeat simple signature-based detection.

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-interactive flows, as signatures would need to be developed for every game (or alternatively for every non-game).

B. ML-based Classification

Figure 1 visualizes 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/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 labeled with class labels corresponding to the network applications or application classes (e.g. interactive and non-interactive) 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. For some algorithms no parameters may be needed. Once the classifier has been trained, new flows can be classified based on their flow characteristics (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 is then used to map network traffic to different QoS classes.

III. Evaluation

In this section we evaluate different machine learning techniques against the requirements of the ANGEL flow classifier. We only consider the requirements that depend on the classification technique. Requirements such as 'data export' are not concerned with how the classification is done. Our evaluation is based on results obtained in previous experiments published in [8] and [7]. These papers also briefly describe the used algorithms.

A. Classification Accuracy

The ANGEL classifier must provide a reasonable high accuracy when differentiating real-time from non real-time flows ($\geq$95 percent). Besides outputting the predicted class for each flow the ANGEL classifier should also compute a measure on how correct the prediction is.

In [8] we have shown that accuracy, recall and precision for separating game from non-game traffic are very high ($\geq$99%) when using the C4.5, Bayes Network or NBTree algorithm both measured on the number of instances and the number of bytes (Bayes Networks only had ($\geq$95%) recall on the bytes though). Naive Bayes with discretisation provides acceptable recall and byte-based precision ($\geq$97%).

Though the Bayes algorithms are not as accurate as algorithms such as C4.5, they are useful when over-fitting is a great concern. Over-fitting means the classifier model built by the ML algorithm is very tuned to the training data and not general enough to classify new instances accurately. Our experience has also shown C4.5 default tree pruning to be very effective in preventing over-fitting.

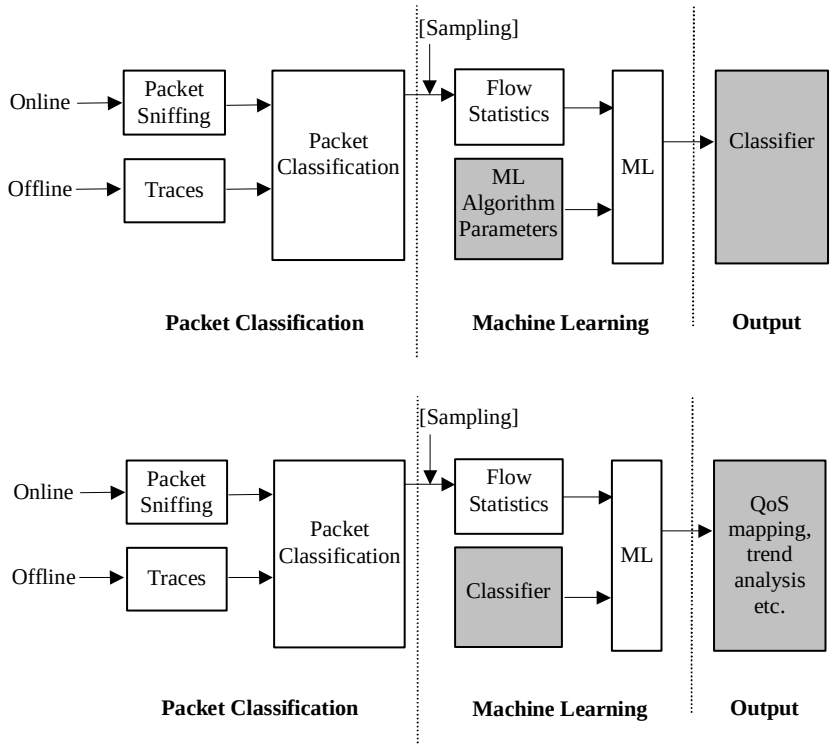

Fig. 1. ML-based flow classification approach: learning phase (top) and classification phase (bottom)

B. Classification Time

The classifier must provide the required accuracy a short time after the flow has started and maintain the accuracy during the flow's lifetime. The time in which an accurate prediction is needed depends on the application e.g. for games this may be longer as joining a game server usually takes some time, but for Voice over IP (VoIP) it may be shorter. In [14] it was shown that classification can occur after seeing as few a 25 packets of a flow - which is a very short time given the inter-arrival time of packets in online game flows. Thus QoS can be provided within a very short period of time.

Besides the actual game flows (traffic exchanged between client and server during the time the client is playing) client-server-based games also exchange a large number of very short probe flows during the process of server discovery (see [8]). In [8] probe flows were included in the dataset, hence we know that both probe flows and game flows can be accurately classified. However, probe flows are problematic as they are so short-lived that QoS rules generated for probe flows would always arrive 'too late' at the ANGEL router. They would not be effective unless the client probes the same server again within the rule timeout limit. Furthermore, as there are usually hundreds up to a few thousand servers available for popular FPS games hundreds or even thousands of rules would be send to the ANGEL router in short time period. This results in a significant amount of bandwidth used by the ANGEL protocol on the bottleneck link and also means possible performance degradation and higher memory consumption on the ANGEL router itself.

It is probably not necessary to provide QoS for probe flows as they are not responsible for a users game play experience. However, there is one benefit in classifying these flows as game traffic. If probe flows are classified as game traffic ping times in the client's server browser would more properly reflect the latency later occurring during game play when ANGEL has detected and prioritised (assuming the access link is congested when probing). Due to the nature of probe flows

C. Flexibility

The ANGEL classifier must be flexible regarding the classification techniques and features used. The ANGEL classifier may use different techniques and or features simultaneously for different flows. The ANGEL classifier may also use different classification techniques and different features simultaneously to determine the class of a single flow.

Basically all machine learning algorithms are flexible in terms of what features can be used. Different machine learning algorithms could be used for different flows at the same time. It is also possible to use different machine learning techniques at the same time on one flow and combine their classification results into a single classification e.g. by using voting schemes.

D. Classification Performance

The ANGEL classifier must be fast enough to classify hundreds of flows per second.

All of the machine learning techniques evaluated in [8] can classify more than 20,000 flows per second. This is pure classification time and excludes overhead such as reading the packet data from a network socket. However, we believe that even when including this overhead classification performance would be suitable. In [7] we have investigated the classification performance of further algorithms. The numbers in both papers cannot be directly compared, because the number of classes and features used for training is different. However, all algorithms evaluated in [7], with the exception of Nearest Neighbour and Naives Bayes with kernel density estimation, are fast enough to classify at least a couple of thousand flows per second.

E. Training Performance

It should not take an unreasonable long time to update/change the classifier. For example, some machine learning algorithms have very long training times, which would prevent reacting quickly to changes in the network traffic mix e.g. if new games with different characteristics appear.

As shown in [8] all algorithms (except NBTree) train fairly quickly and would allow regular updates. NBTree is very slow if the dataset is large in terms of number of instances and/or features and could only be used if very infrequent updates would be sufficient. In [7] we have evaluated training times for a larger number of algorithms. Again the numbers in the two papers cannot be directly compared because the training datasets were different. However, the trends are the same. Also unsuitable for regular updates would be neural networks (which train even slower than NBTree) and the adaboosted C4.5 and Naive Bayes. Support vector machines are also significantly slower than the quick training algorithms C4.5, Bayes Net and Naive Bayes but much faster than NBTree and neural networks.

IV. Conclusions and Future Work

In this paper we have evaluated if machine learning algorithms can be used to separate interactive game traffic from non-interactive traffic under the constraints of the ANGEL architecture requirements for flow classification. We found that games can be separated both from each other, and from other network traffic. With a feature space of 36 flow statistics, overall classification accuracies of greater than 99% were achieved. The results for the byte accuracy are similar - though slightly worse - when compared to the flow-based results. We believe the byte accuracy could be improved however as algorithms optimise classifiers only on the number instances during training. Increased per-byte accuracy may be gained if a classifier is trained with instances given different weights based on flow volume.

Based on the previous evaluation in [8] the best performance was obtained using the C4.5 decision tree and Bayes Network algorithms. Both algorithms provide very high accuracy and are fast. C4.5 provides slightly better accuracy and classification times while Bayes Networks train quicker. The NBTree algorithm provides good accuracy, but has very long training times and could only be used if the classifier does not need to be changed quickly. The Naive Bayes algorithm with discretisation could also be used as it is a simple algorithm with a good trade-off between accuracy and speed.

C4.5 and Bayes Network algorithms have the advantage that the classifier can be interpreted by humans. In case of C4.5 the classifier is a decision tree, that can be translated into if-then rules. In case of Bayes Networks the classifier is a graph where the nodes represent features or classes and weights are conditional probabilities. However, for very complex classifiers human interpretation remains difficult due to the sheer size of the tree or graph.

In terms of ease of use, Bayes Network is the most difficult as there are numerous metrics and search techniques for identifying the network structure, and multiple techniques for estimating the probabilities. Naive Bayes is the most straightforward, as it does not require any parameters and the algorithm is quite simple. C4.5 and NBTree algorithms lie somewhere in between.

In [7] we also evaluated neural networks and support vector machines (SVMs). Neural networks and SVMs did not perform as well as expected. On average they produced accuracies of 10% less than the best algorithms. However, with careful parameter tuning they might be able to provide high accuracies. But param-

eter tuning is cumbersome and may in result in overfitting the classifier to a specific training dataset. Both algorithms have the disadvantage that their classification models lack the ability of being understandable for humans. Especially neural networks are black boxes. Furthermore, evaluation in [7] has shown that both algorithms are not among the fastest. However, SVMs still provide sufficiciently fast classification and training times. Neural networks can classify fairly fast but training is very slow and would prohibit frequent classifier updates.

In [7] we also evaluated the use of a boosting algorithm with C4.5 and Naive Bayes with discretisation. Boosting is a technique that trains multiple 'weak' classifiers and combines them into one efficient. We found that the accuracy increase was only small since both boosted algorithms already perform very good. But boosting severely reduces classification and training speeds. Given the high accuracy measured for separating game traffic from non-game traffic in [8] we think boosting should not be used as it would not significantly increase accuracy but severely slow down classification and training.

We found the most useful features for separating game from non-game traffic were packet length statistics (most importantly the minimum forward packet length), protocol and idle times (see [8]). A similiar dominance of packet length statistics and protocol over other statistics such as inter-arrival times or volume was also observed in [7].

V. Acknowledgements

This work was supported from 2005 to early 2007 by the Smart Internet Technology Cooperative Research Centre, http://www.smartinternet.com.au.

Intellectual content within this report builds on knowledge obtained during an earlier project at CAIA (http://caia.swin.edu.au/urp/dstc), supported 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," in *Proceedings 11th IEEE International Conference on Networks (ICON) 2003*, September 2003.

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

[3] L. Stewart, G. Armitage, P. Branch, S. Zander, "An Architecture for Automated Network Control of QoS over Consumer Broadband Links," in *IEEE TENCON 2005*, November 2005.

[4] S. Zander, "Misclassification of Game Traffic based on Port Numbers: A Case Study using Enemy Territory," CAIA, Tech. Rep., March 2006, http://caia.swin.edu.au/urp/dstc/.

[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," in *IEEE 30th Conference on Local Computer Networks (LCN 2005)*, November 2005.

[7] N. Williams, S. Zander, G. Armitage, "Evaluating Machine Learning Algorithms for Automated Network Application Identification," CAIA, Tech. Rep. 060410B, April 2006, http://caia.swin.edu.au/urp/dstc/.

[8] ——, "Evaluating Machine Learning Methods for Online Game Traffic Identification," CAIA, Tech. Rep. 060410C, April 2006, http://caia.swin.edu.au/urp/dstc/.

[9] J. But, L. Stewart, S. Zander, "ANGEL - Architecture," CAIA, Tech. Rep. 070228A, February 2007.

[10] N.Williams, S. Zander, "Timeliness and Stability of Network Flow Classification using Machine Learning," CAIA, Tech. Rep., March 2006, http://caia.swin.edu.au/urp/dstc/.

[11] "IANA Port Numbers," http://www.iana.org/assignments/port-numbers.

[12] "Ports database," http://www.portsdb.org/.

[13] Cisco Systems, Inc., "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.

[14] T.T.T. Nguyen, G. Armitage, "Training on multiple sub-flows to optimise the use of Machine Learning classifiers in real-world IP networks," in *IEEE 31st Conference on Local Computer Networks*, November 2006.