Accepted 31July06 to be presented at IEEE LCN 2006, November 2006. Paper made possible in part by a grant from the Cisco University Research Program Fund at Community Foundation Silicon Valley

Training on multiple sub-flows to optimise the use of Machine Learning classifiers in real-world IP networks

Thuy T.T. Nguyen, Grenville Armitage

Centre for Advanced Internet Architectures
Swinburne University of Technology, Melbourne, Australia
{tnguyen, garmitage}@swin.edu.au

Abstract

Literature on the use of machine learning (ML) algorithms for classifying IP traffic has relied on full-flows or the first few packets of flows. In contrast, many real-world scenarios require a classification decision well before a flow has finished even if the flow's beginning is lost. This implies classification must be achieved using statistics derived from the most recent N packets taken at any arbitrary point in a flow's lifetime. We propose training the classifier on a combination of short sub-flows (extracted from full-flow examples of the target application's traffic). We demonstrate this optimisation using the Naïve Bayes ML algorithm, and show that our approach results in excellent performance even when classification is initiated mid-way through a flow with windows as small as 25 packets long. We suggest future use of unsupervised ML algorithms to identify optimal sub-flows for training.

1. Introduction

Real-time traffic classification has potential to solve difficult network management problems for Internet service providers (ISPs) and their equipment vendors. Network operators need to know what is flowing over their networks promptly so they can react quickly in support of their various business goals. Traffic classification may be a core part of automated intrusion detection systems [4][5][6], used to detect patterns indicative of denial of service attacks, trigger automated re-allocation of network resources for priority customers [1], or identify customer use of network resources that in some way contravenes the operator's terms of service. More recently, governments are also clarifying ISP obligations with respect to 'lawful interception' (LI) of IP data traffic [3]. Just as telephone companies must support interception of telephone usage, ISPs are increasingly subject to government requests for information on network use by particular individuals at particular points in time. IP traffic classification is an integral part of ISP-based LI solutions.

Commonly deployed IP traffic classification techniques have been based around direct inspection of each packet's contents at some point on the network. Simple classification infers application type by assuming that most applications consistently use 'well known' TCP or UDP port numbers. Packets seen with the same source <address,port>, destination <address,port> and protocol type (TCP or UDP) within a finite period of time are considered to belong to a 'flow', and the flow is associated with a particular application. However, many applications now use random (or at least obscure) port numbers [2]. Consequently, more sophisticated classification techniques infer application type by looking for application-specific data, or well-known protocol behaviour, within the TCP or UDP payloads [7].

Unfortunately, the effectiveness of such 'deep packet inspection' techniques is diminishing. Packet inspection uses two assumptions: third parties unaffiliated with either source or recipient can easily parse each IP packet's payload, and the classifier knows the precise syntax of each application's packet payloads. Two issues undermine the first assumption – customers may use encryption to obfuscate packet contents (including TCP/UDP port numbers), and governments may impose privacy regulations constraining the ability of third parties to lawfully inspect payloads at all. The second assumption imposes a heavy operational load - commercial devices will need repeated updates to stay ahead of regular (or simply gratuitous) changes in every application's packet payload formats.

The research community has responded by investigating classification schemes capable of inferring application-level usage patterns without deep inspection of packet payloads. Newer approaches classify traffic by recognising statistical patterns in

externally observable attributes of the traffic (such as typical packet lengths and inter-packet arrival times). Their ultimate goal is either clustering IP traffic flows into groups that have similar traffic patterns, or classifying one or more applications of interest.

A number of researchers are looking particularly closely at the application of machine learning (ML) techniques to IP traffic classification [8][9][10][11]. Attributes of flows calculated over multiple packets (such as maximum or minimum packet lengths in each direction, flow durations or inter-packet arrival times) are known as 'features'. Classification involves two stages – training the ML algorithm to associate sets of features with known traffic classes (creating rules), and applying the ML algorithm to classify unknown traffic using previously learned rules. Every ML algorithm has a different approach to sorting and prioritising sets of features, which leads to different dynamic behaviours during training and classification.

Our research focuses on the practical application of ML algorithms to traffic classifiers deployed in operational IP networks. Most published research has focussed on the efficacy of different ML algorithms when applied to entire datasets of IP traffic – trained and tested over full flows consisting of thousands of packets and hundreds or thousands of flows. Some newer work has tried classification using the first few packets of a flow. Yet in real networks traffic classifiers must reach decisions well before a flow has finished, they may not see the actual start of a flow, and the application's statistical behaviour may change over the lifetime of each flow. In addition there may be thousands of concurrent flows, and the classifier will operate with finite CPU and memory resources.

In this paper we present a novel modification to traditional ML training and classification techniques that optimises the classification of flows within finite periods of time and with limited physical resources. We propose that realistic ML-based traffic classification tools should:

- Operate the ML classifier using a sliding-window over each flow – the classifier can see (or must use) no more than N packets of a flow at any given time.
- Train the ML classifier using sets of features calculated from multiple sub-flows – each sub-flow is a fragment of N consecutive packets taken from different points within the original application flow's lifetime.

N is chosen to reflect memory limitations in the classifier implementation or the upper bound on the time allowed for classifying a flow. Training on multiple sub-flows allows the sliding window classifier to properly identify an application regardless of where within a flow the classifier begins capturing packets.

We illustrate our proposal's broader benefits by considering an ISP that wishes to automatically and quickly detect online interactive game traffic mingled in amongst regular consumer IP traffic. We apply our modifications to the well-known Naïve Bayes ML algorithm and demonstrate distinct improvements in classification accuracy and timeliness.

Our paper is organised as following. Part II briefly summaries key ML concepts and related work. We describe our problem and proposed approach in part III. Part IV illustrates our proposal and experimental method. We analyse the results in Part V and discuss conclusions and future work in Part VI.

2. Machine learning and related work

First we summarise the basic concepts of ML and review related work applying ML to IP traffic classification.

2.1. Concepts and Terminology

ML classification algorithms all assume that a 'class' of traffic can be identified using statistical analysis of traffic features. A feature may be any numerical attribute calculated over multiple packets – examples include mean packet lengths, standard deviations of inter-packet arrival times, total flow lengths (in bytes and/or packets), and so on. Features calculated over individual flows result in 'feature values' unique to that flow. Not all features are equally useful, so practical classifiers chose the smallest set of features that result in efficient differentiation between members of a class and other traffic outside the class.

ML algorithms can utilise either unsupervised or supervised learning. Unsupervised ML algorithms allocate flows to classes based on clustering of similar feature values. Supervised ML algorithms use examples of IP traffic matching the class of traffic that are later to be identified in the network. The supervised ML algorithm then seeks out traffic flows whose feature values are similar to the traffic on which it was trained [15][16]. A supervised-learning algorithm typically also benefits from being provided with examples of traffic *outside* the class it is being trained to recognise.

Two metrics often used to evaluate ML classification algorithms are *Recall* and *Precision*. If a classifier is trained to identify members of class X:

- Recall is the proportion of class X's instances which are correctly classified as belonging to class X.
- Precision is the proportion of the instances which truly have class X among all those classified as class X.

Both metrics range from 0 (poor) to 100% (optimal). In this paper we utilise both metrics but it is important to note that high Precision only is meaningful when the classifier achieves good Recall.

2.2. Previous related work

McGregor et al. [8] used the Expectation Maximization (EM) algorithm to cluster traffic with similar observable properties into different application types using a fixed set of attributes. The algorithm was found to separate traffic into a small number of basic classes. In [10] we proposed an ML-based approach for identifying different applications using the Autoclass algorithm, which is based on Bayesian classification and the EM algorithm. We studied a wide range of applications and shown that some separation between the different applications can be achieved. In [13] Roughan et al use ML nearest neighbours and linear discriminate analysis approaches to map different network applications to different QoS classes. The flow features used were the average packet size, flow duration, number of bytes per flow, number of packets per flow, interarrival variability and Root Mean Square packet size. Flow features used for the training process are the mean value of all flows per particular applications over a 24-hour period. Moore and Zuev in [9] used the supervised ML Naïve Bayes technique to categorise Internet traffic by application. A large number of flow features (up to 248) had been used to train the classifier. Among those features were flow duration, server port, packet interarrival time, payload size, effective bandwidth based upon entropy and Fourier Transform of the packet interarrival time. The work of Karagiannis et al [26] developed an application classification method based on the behaviours of the source host at the transport layer, which are divided into three different levels. The social level captures and analyses the interactions of the examined host with others hosts in terms of the number of hosts it communicates with. The host's popularity and other hosts in its community's circle are considered. The role of the host in which it acts as a provider or the consumer of a service is classified in the functional level. And finally, transport layer information, such as the 4-tuple of the traffic (including source and destination IP addresses, and source and destination ports), flow characteristics such as the transport protocol and average packet size are used.

Bernaille et al [25] recently proposed a technique using unsupervised ML (Simple K-Means) algorithm that classifies different types of TCP-based applications using the first few packets of the traffic flow. With a small dataset of one-hour trace, their result showed that more than 80% of total flows are correctly identified for a number of applications by using the first five packets of each TCP flow.

3. Problem statement and proposal

Until recently most published work has relied on features calculated over the full lifetime of flows – both for training and for subsequent classification. The efficacy and timeliness of ML classifiers has not been explored under conditions where the flow's beginning is missed and the classifier only sees a subset of a flow's packets. Yet even [25] assumes the initial packets of every flow are captured and available for classification.

In contrast to the previous work, we consider not only the timeliness of a ML traffic classifier, but also its sustainability in performance when monitoring the traffic flows over their lifetime with the constraints of limited physical resources, and when it misses the start of the traffic flows.

Our goal - classification based on only the most recent N packets of a flow (for some small value of N) - is driven by two primary considerations. First, an ML classifier is likely part of a larger system (for example, automated QoS control) that must react swiftly once it identifies a new flow as belonging to a class of interest. Reducing the time taken to detect traffic of interest implies reducing the number of packets that must pass the monitoring point before classification can be achieved. Second, re-calculating features over a sliding window of N packets requires us to buffer the most recent N packets (so we can remove the effect of the Nth most recent packet when we receive a new packet on the same flow). Particularly on high-speed networks a classifier may be observing (tens of) thousands of concurrent flows. Minimising the number of buffered packets per flow provides a beneficial reduction of physical memory requirements.

A classifier also cannot assume it will see the beginning of all flows. For example, classification may be initiated at a point in time when many thousands of flows are already in progress. A classifier should thus be capable of recognising flows using N packets starting from anywhere in a flow.

Using a sliding window of N packets exposes another potential problem. Application flow statistics often change during the lifetime of a flow (for example, the initial handshake of a new SMTP connection may look quite different to the traffic while transferring the body of each email). A classifier trained on feature values calculated over entire flows (as done in most previous research) may not recognise members of the class when presented with feature values calculated over subsets of an unknown flow.

The preceding considerations give rise to our novel proposal for training ML classifiers. First extract two or more sub-flows (of N packets) from every flow that represents the class of traffic we wish to identify in the future. Each sub-flow should be taken from places in the original flow having noticeably different statistical properties (for example, the start and middle of the flow). Each sub-flow would result in a set of instances with feature values derived from its N packets. Then train the ML classifier with the combination of these sub-flows rather than the original full flows.

4. Illustrating our proposal

To illustrate our proposal we construct the following scenario: a real-time Naïve Bayes classifier must accurately identify Wolfenstein Enemy Territory (ET) [12] traffic mixed in amongst thousands of unrelated, interfering traffic flows. ET is a highly interactive online game representative of applications whose traffic characteristics can change significantly over the lifetime of each flow. We compare classification accuracy using full-flows and sub-flows for various values of N, and show that training with multiple sub-flows allows us to achieve high Recall and Precision even for N as small as 25 packets. We show that classification performance can be maintained even when packets are missed at the beginning of a flow.

4.1. The Naïve Bayes algorithm

Naïve Bayes is a well-understood supervised-learning algorithm whose classification approach is based on probabilistic knowledge [15][16]. In this paper we use the Naïve Bayes implementation in WEKA tools [17] with the use of supervised discretisation to process numeric attributes option. (Evaluation of our proposal with other ML algorithms is the subject of current, ongoing work.)

4.2. Flows and features

Traffic flows are bidirectional streams of packets between a given pair of hosts. A flow is defined using the 5-tuple of source and destination IP addresses, protocol (TCP & UDP) and the source and destination ports. The first packet seen by the classifier determines the 'forward' direction. For UDP traffic a flow is considered to have stopped when no more packets are seen for 60 seconds. For TCP traffic, a flow is stopped when the connection is explicitly torn down or no packets are seen for 60 seconds (which ever comes first) [14].

We trained and classified using the following features, calculated separately in the forward and backward directions:

- Inter-packet arrival interval (minimum, maximum, mean and standard deviation)
- Inter-packet length variation (minimum, maximum, mean and standard deviation)
- IP packet length (minimum, maximum, mean and standard deviation)

These features are simple to calculate over a sliding window, are independent of flow length (thus allowing timely calculation before an application has finished) and require no knowledge of packet contents (thus minimizing privacy intrusion or dependency on particular packet payload encoding). We modified Netmate tool [24] to calculate feature values for our analysis.

4.3. Constructing training and testing datasets

To show the effectiveness of our proposed approach we used completely different datasets for training and testing our classifiers. Each dataset consisted of both ET and non-ET traffic because supervised learning algorithms work best when trained with examples of traffic in the class of interest and traffic known to be outside the class of interest ('interfering' traffic).

The ET traffic consisted of two separate month-long traces collected during May and September 2005 at a public ET server in Australia [18]. The distribution of domestic and international traffic on this server was consistent with previously published work [22]. For our interfering (non-ET) traffic we utilised public traces: two 24-hour periods collected by the University of Twente, Germany, on February 6th and 7th 2004 [20]. We will refer to the interfering traffic sources as T1 and T2 respectively.

Raw ET traffic traces taken at an ET server typically contain far more short flows (clients probing the server, usually less than 10 packets in total) than actual game-play flows [22]. Balanced ET datasets for

each month were created by taking all game-play flows and then sampling an equal number of non-game play (probe) flows from the raw monthly traces. Table 1 summarises the resulting balanced ET datasets we then used for training and testing.

Table 1. ET datasets

Data Traces	Total Flows	Total Packets	Total Bytes
May	8688	107M	14G
Sep	6888	187M	26G

Table 2. Sampled interfering application flows

Application	Total flows (T1, T2)	Total flows (T1, T2)
HTTP, HTTPS	13.8K	13.3K
DNS, NTP	2.4K	2.7K
SMTP, IMAP, POP3, Telnet, SSH	0.6K	0.5K
HalfLife	8.7K	10K
Kazaa, Bittorrent, Gnutella, eDonkey	48K	56K

Our interfering traffic datasets were built by extracting flows from T1 and T2 belonging to a range of common applications. As payloads were missing we inferred application type from the port numbers (judged an acceptable approach because our primary criteria for interfering traffic is that it was not ET). For each application's default port(s) we sampled a maximum of 10000 flows per raw tracefile. Table 2 summarises the overall mix of traffic in our resulting interfering datasets.

For each experiment we trained our classifiers using a mix of ET traffic from the May dataset and interfering traffic from T2 (total of 8,688 flows for ET traffic mixed with 82,957 flows in the interference class. Subsequent testing of each classifier scenario was performed using a mix of ET traffic (6,888 flows) from September and interfering traffic from T1 (a mixture of 73,672 flows).

4.4. Some statistical properties of ET traffic

Consistent with many other online first-person shooter games, ET traffic seen at a server can exhibit three different phases: clients probing the server, clients connecting to the server and clients actually playing a game on the server [23].

Figure 1 partially illustrates this variation of an ET flow's characteristics as a scatter plot of two features - Standard Deviation versus Mean of packet length - calculated with N = 25 across 1000 samples of the May dataset. Full flow and In-game feature values are shown on the left, client probing and client connecting feature

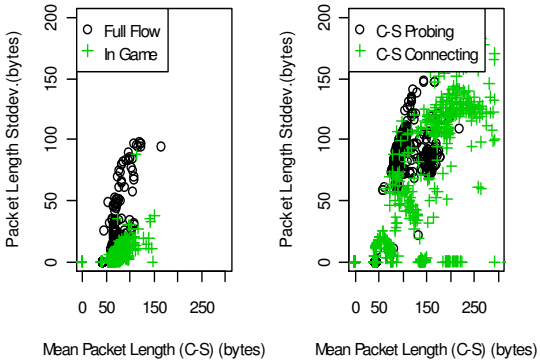

Figure 1. Packet Length from Client to Server

values are shown on the right. With only two features the regions are partially overlapping and partially disjoint. A similar mix of overlapping and disjoint regions also occurs with other features (such as Inter-packet arrival time and Inter-packet length variation). Figure 1 tells us that a classifier trained on full flow feature values may have trouble recognising the clusters of feature values calculated over small windows of packets.

5. Results and analysis

First we look at the effectiveness of classifying data using a sliding window across the test dataset and an ML classifier trained on full-flow training sets. Then we show how Recall and Precision improve significantly when each ML classifier is trained using multiple sub-flows instead. We use windows of size N = 10, 25, 50, 100, 500 and 1000 packets. (During ET game-play we see 20 pps from server to client and roughly 28 pps from client to server, so these windows correspond to 0.2, 0.5, 1.0, 2.1, 10.4 and 20.8 seconds of actual time). Recall and Precision results are averaged across 6,888 ET flows and 73,672 interfering flows in the test dataset.

5.1. Training on full-flows, classifying with a sliding window

Figure 2 shows Recall and Precision for Naïve Bayes as each sliding window moves across the test dataset. M is the number of packets 'missed' from the beginning of each flow in the test dataset. The graphs covers two periods – early client contact with the game server (0 <= M <= 90) and during active game-play (1000 <= M <= 9000). The classifier has been trained

using features calculated over full flows from the training datasets.

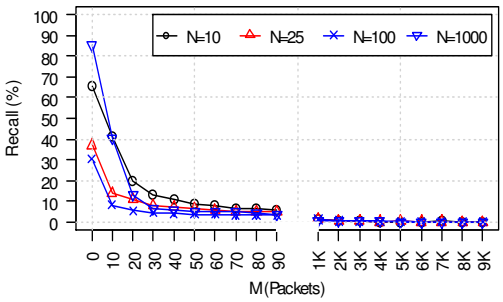

Figure 2. ET Recall: Classifier trained with full flows, tested with four different sliding windows

The classifier's Recall degrades rapidly as we move further from the start of each flow. Recall with small windows (N = 10 or 25) is poor (65% and 40%) at the flow's beginning and drops off rapidly if we miss the start of the flow. Recall with N = 1000 is decent (85%) when the flow is captured from the beginning, but rapidly drops below 10% if classifier misses more than the first 30 packets. The Precision is 100% for all window sizes and M values when Recall > 0. Classifying in the middle of game-play (M > 1000) gives Recall close to 0%, making the high achieved Precision somewhat meaningless.

5.2. Training with individual sub-flows

Figure 3 shows the impact on Recall when the Naïve Bayes classifier is trained using features from 25 packet sub-flows rather than full flows. Five separate variants of the classifier are trained, using sub-flows that cover packets 1-25, 21-45, 41-65, 61-85 and 2001-2025 respectively of the original full flows in the training datasets. In each case the classifier is tested using a sliding window of 25 packets.

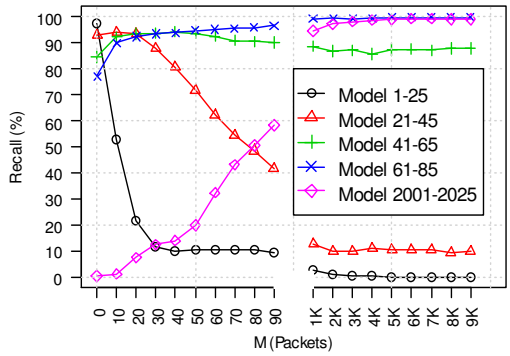

Figure 3. ET Recall: Classifier trained on 25-packet sub-flows, N = 25 for classification

Recall drops off quickly if we miss more than the first 10 packets of a flow when trained on packets 1-25. On the other hand, Recall stays low until the sliding window has moved beyond the early period of each flow (M >>= 90) when trained on packets 2001-2025. When trained on packets 21-45 the classifier's Recall is quite good even if we miss 30 or 40 packets, but eventually becomes quite poor. Only by training on packet 41 onwards (e.g. Model 41-65, and so on) can the classifier exhibit good Recall when exposed to game-play traffic (M > 90). Precision remained from 97.5% to 99.5% for Recall above 50% for each model.

Compared to Figure 2 training on a sub-flow picked from within each original training flow significantly improves our classification performance for M > 0 (i.e. real-world scenarios where the classifier cannot be sure it sees the start of every flow).

5.3. Training with multiple sub-flows

A logical next step is to train the classifier on a combination of multiple sub-flows instances representing different time periods within the original full-flows. The classifier will then recognise new flows if they have statistical properties similar to any of the sub-flows on which the classifier was trained.

To illustrate this idea we trained a classifier using the combination of different sub-flows at the same time. From several tests with combinations of two, three, four different sub-flows, we found the combination of Model 1-25, 21-45, 41-65 and Model 2001-2025 from Figure 3 produces excellent classification results.

Figure 4 shows this new classifier's Recall as a function of M ('Multi Sub-Flow Model N=25'), along with Recall for a classifier trained on Model 61-85 (best performed model in Figure 3) (' Best Single Sub-Flow Model N=25') and a classifier trained on full-flows (using sliding windows of N=1000 and N=25).

The multiple sub-flows curve shows excellent Recall early in a flow's life (M < 40) (97.7-99% compared to 77.1-93.4% for single sub-flow model). For M > 40 the Recall is much the same as training on the single sub-flow (95.1-99.6% vs. 93.4-99.7% respectively). Training the classifier on full flows leads to substantially degraded Recall relative to training on sub-flows. Precision held steady at 98% when trained on the multiple sub-flows (similar to the single sub-flow model).

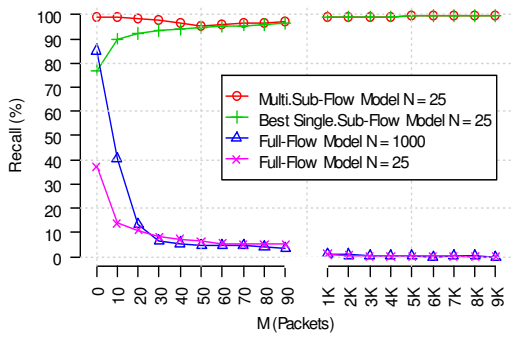

Figure 4. Recall: Comparing full-flow and sub-flow training of the Classifier

The key point of Figure 3 and Figure 4 is that for applications with time-varying traffic characteristics there are significant benefits to training ML classifiers using features calculated over one (or more) sub-flows rather than full-flows.

It is worth noting that we performed similar comparisons using N=10, but the absolute Recall and Precision rates were lower than for N=25. Depending on the particular application you are trying to classify, there will be a trade-off between keeping N low (for timely classification and reduced memory consumption) and keeping N high (for acceptable Recall and Precision). Characterising this trade-off is a subject of further research.

5.4. Future work: Selecting sub-flows using unsupervised ML algorithms

Our preceding analysis depended on manual inspection of ET's particular traffic characteristics. Training a classifier for optimal recognition of another application may require entirely different choice of sub-flows. Ideally we would like to avoid having to manually inspect and identify the optimal set of sub-flows for each application of interest.

There is a potentially fruitful area of future work along these lines. We propose utilising unsupervised classification ML algorithms to automatically identify key sub-flows within examples of an application's full flows. Intuitively this seems reasonable – unsupervised learning algorithms identify 'natural' clustering of sub-flows [16], from which we may identify a set sub-flows representing key statistical characteristics of the full flow. (The existence of natural clustering of feature values in ET was hinted at in Figure 1.)

We ran a preliminary version of this idea on our ET traces (using the Simple Expectation Maximisation (EM) Clustering ML algorithm [27] implemented in WEKA [17]) and found eight natural clusters of 25-packet sub-flows during a game's full flow. From this we obtained eight sub-flows and trained and tested our Naïve Bayes classifier. Compared against the use of four sub-flows in Figure 4 the eight sub-flows achieved slightly better Recall and basically identical Precision.

This brief analysis suggests that manual selection of sub-flows for training is not necessary in the general case. However, there are trade-offs. Initial investigation suggests that whilst Recall increases with additional sub-flows the Precision degrades somewhat. We intend to further develop this line of research, identifying rules and unsupervised learning algorithms that are best suited for the automatic generation of multiple sub-flows with which to train supervised ML classifiers. (Nevertheless, practical classifiers can still be trained using manually identified sub-flows in the absence of a well developed method of automatic sub-flow identification.)

6. Conclusions

Practical real-time traffic classifiers must accurately classify traffic in the face of a number of constraints:

- The classifier should use statistical methods (such as ML algorithms) as TCP/UDP port numbers may be misleading, and packet payloads may be opaque against direct interpretation
- ML classification should be done over a small sliding window of the last N packets (to keep memory requirements down and perform classification in a timely manner)
- The classifier must recognise flows already in progress (the flow's beginning may be missed)
- Application's can change their network traffic patterns over time.

However, most previous research has focused on training ML classifiers using statistical features calculated across entire flows. This leads to poor performance of the classifier when a flow's early packets are missed or the classifier is using a restricted sliding window. Some recent literature discusses classification on only the first few packets of every flow, but they are also unable to cope when those early packets are missed.

We propose a novel solution: The ML classifier should be trained using statistical features calculated over multiple short sub-flows extracted from full flows generated by the target application. The sub-flows are picked from regions of the application's full flows that have noticeably different statistical characteristics.

We show that this can significantly improve a classifier's performance when using a small sliding

window, regardless of how many packets are missed from each flow's beginning. Our proposal is illustrated by constructing, training and testing a Naïve Bayes classifier for the detection of Wolfenstein Enemy Territory online game traffic. With this particular scenario we saw excellent results when trained on four to eight sub-flows and using a sliding window of only 25 packets.

A number of future research directions open up. These include:

- The use of unsupervised ML algorithms to automatically identify optimal sets of sub-flows to use when training the main classifier
- Characterising the optimal sliding classification window size (N) for a wider range of applications
- Identifying how varying N trades classification performance (Recall and Precision) against resource consumption (memory consumption in the classifier and timeliness of classification)
- Demonstrating the utility of our proposal for other ML algorithms (not just Naïve Bayes)
- Evaluating the impact on classification accuracy of packet loss in real networks (no literature has explored this aspect to date)
- Exploring further reduction in the number of features that must be calculated in real-time to still achieve acceptable performance
- Testing our proposal in the presence of a larger and more diverse collection of interfering traffic.

Overall we believe this small proposal significantly assists the use of ML algorithms inside practical and deployable IP traffic classifiers.

References

[1] 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.

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

[3] F. Baker, B. Foster, C. Sharp, "Cisco Architecture for Lawful Intercept in IP Networks," Internet Engineering Task Force, RFC 3924, October 2004

[4] "Snort - the de facto standard for intrusion detection/prevention," http://www.snort.org (accessed April 21, 2006)

[5] "Bro Intrusion Detection System - Bro Overview," http://bro-ids.org (accessed April 21, 2006)

[6] V. Paxson, Bro: A System for Detecting Network Intruders in Real-Time, Computer Networks, 31(23-24), pp. 2435-2463, 14 Dec. 1999

[7] S. Sen, O. Spatscheck, D. Wang, "Accurate, Scalable In-Network Identification of P2P Traffic Using Application Signatures", WWW 2004, New York, USA, May 2004

[8] 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

[9] Denis Zuev, Andrew Moore, "Internet traffic classification using bayesian analysis techniques", ACM SIGMETRICS 2005, Banff, Canada, June, 2005

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

[11] S. Zander, N. Williams, G. Armitage, "Internet Archeology: Estimating Individual Application Trends in Incomplete Historic Traffic Traces", Passive and Active Measurement Workshop (PAM 2006), Adelaide, Australia, March 30 - 31, 2006

[12] "Wolfenstein," http://games.activision.com/games/wolfenstein (viewed 19 December 2005)

[13] M. Roughan, S. Sen, O. Spatcheck, and N.Duffield, Class-of-Service Mapping for QoS: A Statistical Signature-based Approach to IP Traffic Classification. In ACM/SIGCOMM IMC, November 2004.

[14] N.Williams, S.Zander, G.Armitage, "Evaluating Machine Learning Methods for Online Game Traffic Identification," CAIA Technical Report 060410C, April 2006

[15] John, G., & Langley, P. (1995). Estimating continuous distributions in Bayesian classifiers. Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence (pp. 338-345).

[16] I. Witten and E. Frank, Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations, Morgan Kaufmann Publishers, 2000.

[17] WEKA 3.4.4, http://www.cs.waikato.ac.nz/ml/weka/ (as of February 2006).

[18] http://gs.act.grangenet.net/ as of 27th April 2006

[19] Auckland-VI trace archive, http://pma.nlanr.net/Traces/long/auck6 .html, as of 26th March 2006

[20] University of Twente - Traffic Measurement Data Repository, http://m2c-a.cs.utwente.nl/repository/, as of 26th March 2006

[21] J. Bussiere, S. Zander, "Enemy Territory Traffic Analysis," (pdf) CAIA Technical Report 060203A, February 2006

[22] 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

[23] G.Armitage, M.Claypool, P.Branch, "Networking and Online Games - Undertanding and Engineering Multiplayer Internet Games," John Wiley & Sons, UK, April 2006 (ISBN: 0470018577)

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

[25] L.Bernaille, R. Teixeira, I. Akodkenou, A. Soule and K. Salamatian, "Traffic Classification On The Fly", ACM SIGCOMM Computer Communication Review, Vol. 36, No. 2, April 2006.

[26] T. Karagiannis, K. Papagiannaki, M. Faloutsos, BLINC: Multilevel Traffic Classification in the Dark, SIGCOMM'05, Aug. 2005

[27] A. Dempster, N. Laird, D. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm, Journal of Royal Statistical Society, Series B, Vol. 30, No. 1, 1977.