Improving Machine Learning Network Traffic Classification with Payload-based Features

Michal Scigocki*, Sebastian Zander
Centre for Advanced Internet Architectures, Technical Report 131120A
Swinburne University of Technology
Melbourne, Australia
6155553@swin.edu.au, szander@swin.edu.au

***Abstract*—Network traffic classification is important in modern computer networks, for example for quality of service management systems that prioritise traffic depending on its class. Using only UDP/TCP port numbers for traffic classification is unreliable. Using payload inspection provides good accuracy if payload can be accessed, however payload-based classification is often slow and the update of signatures is cumbersome. In recent years researchers have focused on developing classifiers that use Machine Learning (ML) techniques to classify traffic based on features, such as packet length statistics. This report explores the use of raw packet payload data as features in ML based network traffic classification. We describe the implementation of payload based features for an existing ML-based network traffic classifier [1], and the experimentation performed to evaluate their performance. Preliminary results show that the new payload-based features or a combination of payload and packet-length features performs better than using packet-length features alone.**

***Index Terms*—Network Traffic Classification, Machine Learning, Payload**

I. Introduction

Network traffic classification is important in modern computer networks for network security monitoring, lawful interception, and quality of service management systems. In the past dedicated port numbers were used to identify network traffic, for example port 80 for HTTP. Today many applications use dynamic port numbers, since they either do not require a dedicated port, or they are trying to avoid classification by using randomised port numbers. Because of this, using only port numbers for network traffic classification is unreliable for many new applications.

Using payload inspection provides good accuracy if payload can be accessed, however payload-based classification is often slow and the update of signatures is cumbersome. Some researchers proposed to use automatically derived packet-payload-based features and showed promising results [2]–[4], but the previous approaches still have limitations. The approaches in [2], [3] require a significant amount of memory and processing time per network traffic flow, which makes them impractical to implement on real routers, especially low-cost home routers with small memory. The approach in [4] limits its memory footprint by inspecting the first 12 payload bytes of packets only, which effectively limits the discriminative power of the classifier.

In recent years researchers have focused on developing classifiers that use Machine Learning (ML) techniques to classify traffic based on features [5], [6]. ML uses large sets of calculated feature statistics to create a classification model based on those feature statistics. For example, an already implemented feature for ML-based network traffic classification is packet length statistics for a network traffic flow, for example the minimum, mean, maximum, standard deviation of the packet sizes. Given the characteristics of these calculated statistics, ML techniques can create a model that can classify traffic into a particular class. For example, traffic flows with short packet lengths can be classified as real-time traffic, and traffic flows with long packet lengths can be classified as file-transfers. While ML classifiers have shown good efficiency and promising accuracy, accuracy is often lower than that of payload-based classifiers (for traffic for which payload signatures exist).

A classifier that uses both statistical features and payload features may yield the best accuracy (if payload can be accessed), however the payload-feature implementation must be more efficient than previous approaches to be practical. Our motivation for this work was to

*The work described in this report was done during the author's winter internship at CAIA in 2013.

develop efficient payload-based features and investigate the accuracy and complexity of payload-based features alone, statistical features alone, and of a combination of payload-based features and statistical features.

Rather than calculating complex features based on payload data, we focus on using the raw payload bytes as features for the ML algorithm. This approach is very efficient and has the potential to yield good results, since any application-layer protocol has very characteristic header structures. For example, the HyperText Transfer Protocol (HTTP), an ASCII based protocol, sends very particularly formatted text strings, for instance GET, POST request headers in the HTTP requests or the status line of the HTTP response [7].

Initially we extract a larger number of payload bytes as features from the traffic, and then an ML technique can be used to automatically reduce the number of features based on the applications to be classified. In particular, we use the C4.5 decision tree classifier [8] because it previously performed well for network traffic classification [9], in terms of accuracy and speed, and it automatically selects the "best" features.

We first implemented payload-based features for the existing DIFFUSE ML-based network traffic classifier [1]. Then we conducted experiments with the C4.5 classifier [10] comparing the accuracy (based on class precision and recall) and classifier complexity (resulting model tree size) of the new payload features, existing packet length features, and the combination of both types of features used together. Our preliminary results are based on a small set of trace files from actual network traffic. A more comprehensive analysis based on a larger set of traces remains future work.

Our findings show that overall the payload features outperform the packet length features in both accuracy, by up to 11%, and classifier complexity, which was approximately halved. The accuracy and classifier complexity with the combined features is very similar to the performance of the payload feature alone.

The report is organised as follows. In Section II we first describe the implementation of the payload-based features and how they are integrated into the existing DIFFUSE ML network traffic classifier. In Section III we describe the details of the experiments conducted, and in Section IV we present the results of the experiments. Section V concludes and outlines future work.

II. Payload Feature Implementation

In this section we will briefly describe the design of DIFFUSE relative to our purposes, and then continue to describe in more detail our implementation of the payload feature. DIFFUSE was developed as a very modular application, allowing extensions for it to be added with relative ease. DIFFUSE's code base is made up of two major parts: the kernel part and the userspace part. The kernel part of a feature module contains the actual logical implementation of the feature, which extracts the feature statistics from the network traffic. The userspace part is for the interaction between user and the application, which allows the user to configure which feature modules to use, and their options (e.g. window-size, partial-windows, etc). Full details of the design of DIFFUSE can be found in the technical report [11].

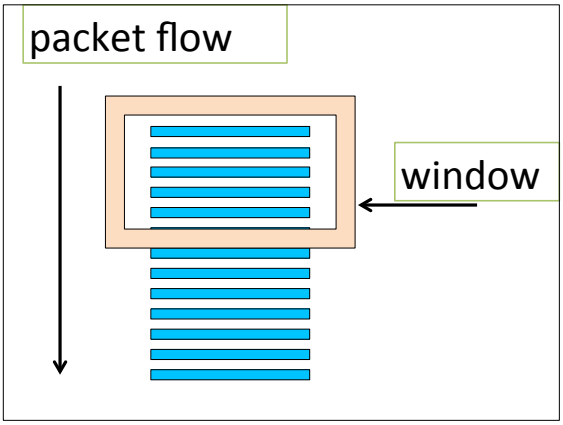

Figure 1. Packet flow window

We now briefly explain the concept of packet windows for ML traffic classification, introduced by Nguyen and Armitage [12]. Traffic flows can be very long and it would be impossible to calculate statistics for all packets in a flow for flows that need to be classified timely. For this reason, we use a moving window of packets within a flow, as shown in Figure 1. Such a window is also called a *sub-flow* [6], [12]. In DIFFUSE, the window size setting determines the amount of packets we use per window. By default statistics are not calculated until a window is completely filled with packets. But this then means that packet flows that are less than the size of the window will not have any statistics calculated for them (e.g. if the window size is 15, flows of 14 packets or less cannot be classified). For this reason, DIFUSE has a partial-windows option, which enables statistics to be calculated for flows shorter than the window size.

While DIFFUSE was developed for real time traffic classification, it also includes an offline testing userspace

tool, ipfw_fstats, which implements the feature extraction functionality for previously captured network traffic traces (currently only for tcpdump capture format). ipfw_stats uses the same feature modules from the kernel code that are used for online classification, and the same userspace code for configuring the feature modules.

The implemented payload based feature (pldbd) for DIFFUSE is based on the existing packet length feature plenbd. The goal of the pldbd feature's implementation is to capture the first 20 payload bytes in each direction for each traffic flow, where a flow is a single bidirectional communication between two end nodes characterised by a 5-tuple (source IP, source port, destination IP, destination port and protocol). For TCP the feature currently collects statistics on a per flow basis (once per flow from the start of the flow), whereas for UDP traffic it collects statistics on a sub-flow basis. We capture the payload in this way because at this stage we are mainly interested in the application protocol header information for the flow. For TCP, a connection-oriented protocol, we expect that information to be transferred at the very start of a flow, whereas for UDP, a connectionless protocol, that information needs to be transmitted in every packet at the beginning of the payload.

A. *Initial setup*

To add a new feature to DIFFUSE, first it has to be installed. DIFFUSE is available for download from the CAIA website [1]. DIFFUSE is available for FreeBSD and Linux systems, and the installation requires kernel source code to be available. The new feature was developed on the PC-BSD 9.1 (FreeBSD 9.1-RELEASE-p4) operating system. DIFFUSE is not currently available for that version of the FreeBSD, so the source code for the latest available version of DIFFUSE was obtained (FreeBSD-9.0_CURRENT_r223644), and only the user-space tools were installed, without recompiling the kernel. Since we were only performing offline experiments on captured trace files, not live traffic classification, we did not need DIFFUSE compiled into the kernel. However, since the offline user-space tools use the same kernel feature implementation code, the new feature should work for live classification as well, but testing this was beyond the scope of the experiments.

B. *diffuse_feature_pldbd.h*

The diffuse feature header file is used to define configuration options used specifically by the feature. These include default window size, a structure to store user configurable options, such as window size and types (partial and/or jumping). The file also includes definitions of the main properties of the feature, such as feature name, feature direction and feature statistics information. The payload feature is a bi-directional feature, which means it calculates statistics in both the forward and backward direction of a flow. The direction of the first packet of a flow is considered to be the forward direction, and the opposite direction is the backward direction. The payload feature contains a total of 40 statistics. Each statistic is the integer representation of a packet payload byte, for the first 20 bytes in the forward direction (fbyte1, fbyte2, fbyte3, ..., fbyte20) and the first 20 bytes in the backward direction (bbyte1, etc). We keep track of the total number of statistics, and the number of forward and backward bytes separately to allow for future extensions of the feature module, such as statistics on the entropy and byte ranges of the payload data.

C. *diffuse_feature_pldbd.c*

The pldbd implementation is based on the existing plenbd implementation. The main difference is in the implementation of the pldbd_update_stats function, where the forward and backward bytes are stored for flows. Some minor differences are present due to the different storage data structure; we are now capturing payload bytes, and not packet length statistics, which in turn has required small modifications to the initialisation, destruction, resetting, and obtaining of the payload statistics.

Storing the payload bytes occurs differently for TCP and UDP flows. For a TCP flow in the forward direction, we first identify the first packet in the TCP flow by checking if a packet's TCP flags are set to SYN only. We store the SYN packet's TCP sequence number, because we know that the packet with the next sequence number (incremented by one) will contain some payload data.[1] Similarly, for the backwards direction we identify the packet with the SYN and ACK TCP flags set, and we store its acknowledgment sequence number. This SYN/ACK packet does not contain any payload data, but the next packet that has an acknowledgment sequence number incremented by one, will be the first backward packet from which we capture the payload.

Capturing the application header data for UDP flows is much simpler. Because UDP is a connectionless pro-

[1]Currently, the very first forward packet for which payload is captured is the last packet of the TCP handshake (ACK packet), which has the same sequence number as the SYN packet and typically has no payload. This is not a major issue, since statistics for packets without payload default to -1. However, in the future this could be improved by ensuring the first forward packet for which we capture payload has the correct sequence number *and is not an ACK packet*.

tocol, application header information is usually sent in the payload of every packet in a flow. This means that we do not need to identify the first payload packet for a flow (as we did for TCP), rather we can simply capture the payload for every packet in a sub-flow.

D. Additional Changes

In order for the new feature to compile, several files needed minor changes that included references to the new pldbd feature. In addition a user-space implementation of the pldbd feature needed to be created (to parse configuration options). Below is a complete summary of all files created and modified for the implementation of the pldbd feature into DIFFUSE:

- Created: sys/netinet/ipfw/diffuse_feature_pldbd.h
- Created: sys/netinet/ipfw/diffuse_feature_pldbd.c
- Modified: sys/modules/Makefile
- Created: sbin/ipfw/feature_pldbd.h
- Created: sbin/ipfw/feature_pldbd.c
- Modified: sbin/ipfw/diffuse.c
- Modified: sbin/ipfw/Makefile
- Modified: sbin/ipfw/ipfw_fstat/ipfw_fstat.h
- Modified: sbin/ipfw/ipfw_fstat/ipfw_fstat.c
- Modified: sbin/ipfw/ipfw_fstat/Makefile

III. Experimental Setup

We now describe how the experiments were carried out. In addition to DIFFUSE, we installed WEKA 3: Data Mining Software in Java [10] to conduct the experiments.[2] We used DIFFUSE to generate Attribute-Relation File Format (ARFF) files, which include all the feature statistics for the traffic from the trace data, and then used WEKA to generate ML models from the ARFF files and perform cross-validation testing.

A. Trace Files

An issue with any network classification experiment is obtaining representative network traffic data. This is even more difficult when the payload of the traffic data is also required. We captured all traffic from a single desktop computer over 19 days of regular use. From those captured trace files, 8 different classes of traffic were observed to be most common. The capture traces had very little FTP traffic, and no long UDP flows (only short DNS and NTP flows), so in addition we used the publicly available traces listed below:

[2]We did not apply the DIFFUSE patch to WEKA, because it is only required when DIFFUSE is used for online traffic classification or for cross-validation in user-space. However, we used WEKA for cross-validation.

Table I
Class instances (sub-flows)

Class	No. of Instances
DNS	18,305
FTP	13,946
HTTP	17,485
IPP	15,892
NTP	17,703
Other	30,619
Quake3	11,163
Skype	10,008
SSH	9,358
HTTPS	16,839
Total	161,325

- FTP: Lawrence Berkeley National Laboratory [13], [14];
- Skype: Polytechnic University of Turin [15], [16];
- Quake 3: Centre for Advanced Internet Architectures [17].

The class instances represent sub-flows for each protocol. The traffic classes we set out to identify were Domain Name System protocol (DNS, UDP port 53), File Transfer Protocol (FTP, TCP port 23), HyperText Transfer Protocol (HTTP, TCP port 80), Internet Printing Protocol (IPP, TCP port 631), Network Time Protocol (NTP, UDP port 123), Quake3 online game traffic (UDP port 27960), Skype traffic (UDP ports: 24762, 42467), Secure Shell protocol (SSH, TCP port 22), HyperText Transfer Protocol Secure (HTTPS, TCP port 443) and Other. The Other class represents sub-flows of other traffic that was captured, excluding the specific traffic classes mentioned before. This small data set is a limitation of this work, and further experiments will be needed with larger, more representative data sets to verify the results. Table I shows the classes and number of instances used for training and testing.

B. DIFFUSE Options

We ran experiments for the following feature sets:

- pldbd only,
- plenbd only,
- pldbd and plenbd together.

For each feature set, we ran multiple experiments for different sliding window sizes (5, 10, ... ,30), with and without partial windows, to verify that results obtained are consistent.

```
./get_fstats tlist.txt <features>
        fopts.txt

java -Xmx4196M -cp weka.jar
        weka.classifiers.trees.J48
        -t <.arff-file> -i -M 100 -x 2
        -d <ouput-model> > mbuild.txt
```

Figure 2. Commands to run experiments

C. WEKA Options

For generating the model and results with WEKA, the C4.5 ML algorithm was used (named J48 under WEKA). We set the minimum number of instances per leaf (option -M) to 100, which resulted in tree sizes of less than 100 nodes, as opposed to trees with thousands of nodes with the default settings. Using more aggressive tree pruning avoids over-fitting the classifier model and also makes the resulting trees more easy to interpret. The experiment conducted was a two-fold cross-validation experiment (option -x), where half the number of the instances of each class are taken to train (create a model), and the other half of the instances is used to test the generated model. Additionally, the option -i was used to automatically generate per class statistics, including precision and recall.

Figure 2 shows the commands used to extract the features from the traffic data and perform the cross-validation testing with WEKA.

IV. Results

Overall, using the packet payload bytes directly as features for ML classification shows promising results in comparison to using only the existing packet length features. The presented results are for a single experiment (with window-size set to 15), but they are representative of the results seen in the other experiments. We show the results for a partial-window experiment specifically, because without that option set, we do not get any results for DNS, and NTP classes, because of their very short flow lengths (usually two packets). In Figure 3 and Figure 4 we see that in almost every case, packet payload features outperform packet length features in terms of precision and recall. Overall we observed an increase of up to 11% in precision and recall. Only in the case of the precision of the IPP protocol, we find packet length performs better. This could be due to a number of factors, such as a non-representative data-set (our IPP data includes traffic to only one printer connected to one print server) or the similarities between IPP and HTTP payload data. A more thorough analysis remains future work.

In Figure 5 we see that payload features produce a smaller tree size than packet length features, in terms of the number of nodes and leaves. The decision tree for payload features is only half the size of the corresponding packet length tree. With a less complex decision tree, the classification of network traffic flows can be completed in fewer steps. Furthermore, with a less complex decision tree packet payload based features produce higher precision and recall overall than packet length features achieve with a more complex decision tree.

It is important to investigate which payload bytes the C4.5 classifier selected to make its classification decisions. We will only briefly examine the case for the HTTP protocol and leave a more thorough investigation as future work. For the payload-only experiment, we find the decisions for HTTP are primarily based on two tests in the decision tree, summarised in Figure 6. The feature test using bbyte16 classifies the majority of instances; it decides that the flow is a HTTP flow if the 16th byte in the backwards direction has an integer value higher than 12. The ASCII character 13 is a “Carriage Return” character. If no error occurred a typical HTTP server response starts with the characters “HTTP/1.1 200 OK”, which is 15 characters (or bytes) long, at the end of which a Carriage Return (the 16th byte) is present, followed by a Line Feed (the 17th byte). Similarly, for the feature test using bbyte9 we find ASCII character 32 to be a “Space” character, which also fits the typical response. Of course HTTP server responses differ, which is why we see that the bbyte9 test has 28 misclassified occurrences (the number after the dash). Overall, we find the classifier’s decisions based on which bytes to decide are logical and relate to the formats of application protocol headers of the various classes.

The findings of the comparison of packet payload features against the combined packet payload and length features are inconclusive. We find the accuracy, in terms of precision and recall, is generally within 1% of each other, and the tree sizes are also relatively similar. Further experimentation and a larger, more representative data set would be required to identify if there is any real benefit of using the combined features.

V. Conclusions

We implemented payload-based features for the existing DIFFUSE ML-based network traffic classifier [1].

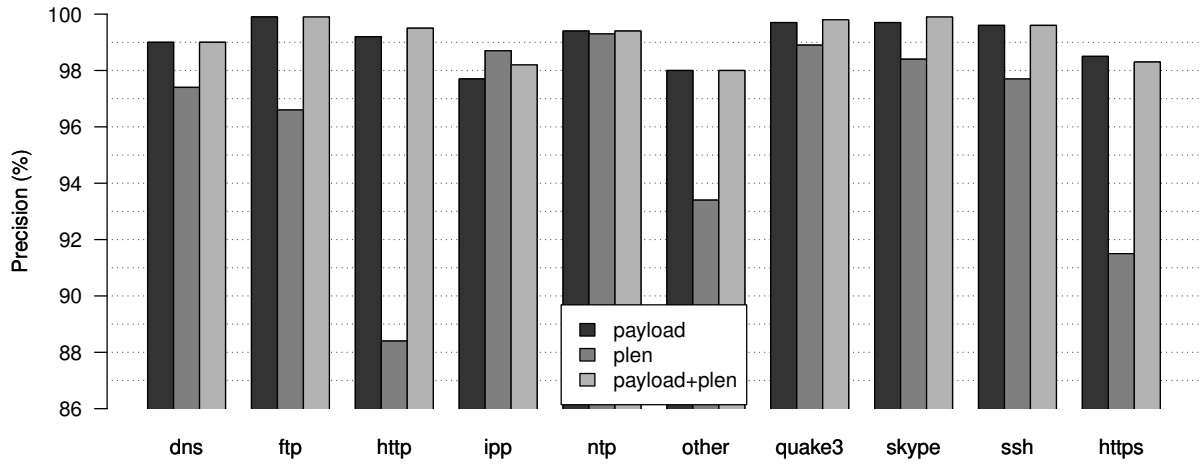

Figure 3. Class precision (window-size=15, partial-windows=yes)

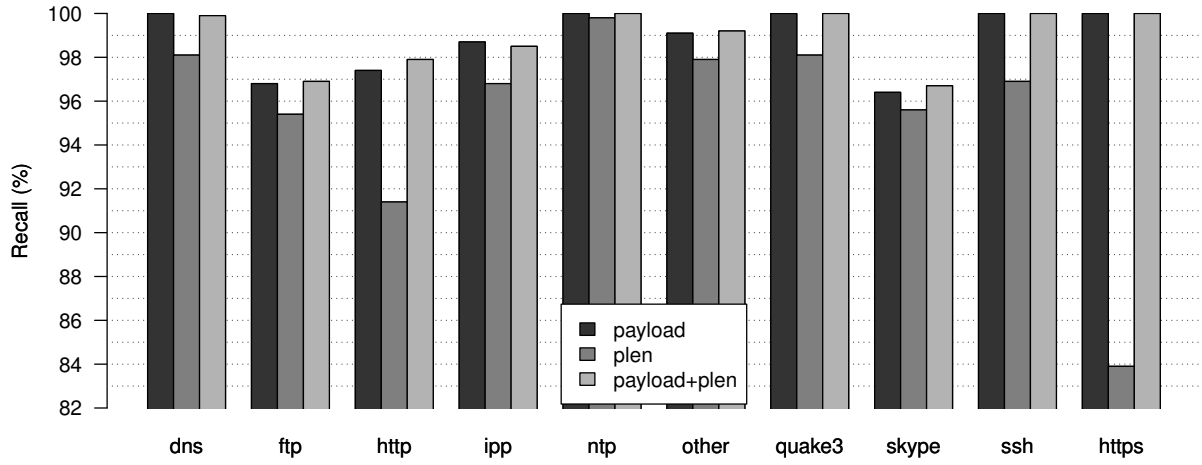

Figure 4. Class recall (window-size=15, partial-windows=yes)

The performance of using raw payload data as features for an ML classifier shows promising preliminary results. The newly implemented payload-based features outperformed the existing packet-length features by up to 11% in terms of class accuracy, with only half the classifier complexity. The combined payload and packet length features resulted in an accuracy that was similar (within 1%) to using only the payload features, and the tree sizes were also relatively similar.

While these preliminary findings show promising results, further experiments with larger, more representative data sets are required to confirm the current findings. We also aim to increase the number of classes in the future. Our current payload features are based on payload from the start of flows (e.g. the start of TCP sessions). In the future we will extend our implementation, so that not only payload from the flow start is considered, but also payload from the last few observed packets can be used.

REFERENCES

[1] S. Zander, "DIstributed Firewall and Flow-shaper Using Statistical Evidence (DIFFUSE)." http://caia.swin.edu.au/urp/diffuse/, Apr. 2012. Accessed: 2013-08-09.

[2] P. Haffner, S. Sen, O. Spatscheck, and D. Wang, "ACAS: Automated Construction of Application Signatures," in *ACM SIGCOMM MineNet Workshop*, (Philadelphia, PA, USA: ACM), Aug. 2005.

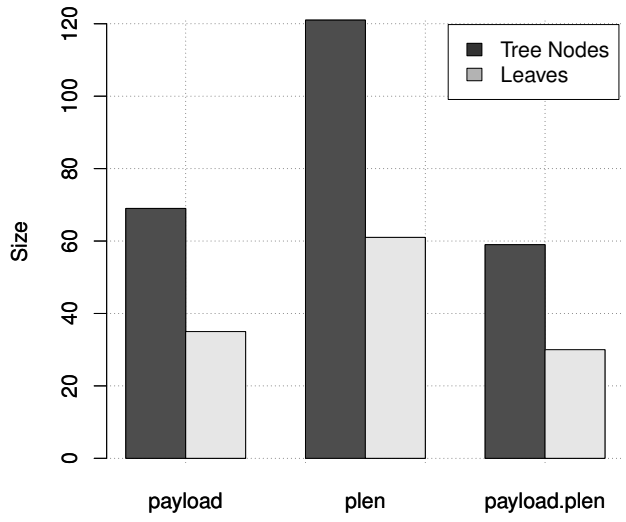

Figure 5. Classifier complexity: feature tree size (window-size=15, partial-windows=yes)

```
bbyte16.pldbd > 12: http (17078.0)
bbyte9.pldbd > 31: http (127.0/28.0)
```

Figure 6. Decision tree entries for HTTP

[3] J. Ma, K. Levchenko, C. Kreibich, S. Savage, G. M. Voelker, "Unexpected Means of Protocol Inference," in *6th ACM SIGCOMM Conference on Internet Measurement (IMC)*, pp. 313–326, 2006.

[4] A. Finamore, M. Mellia, M. Meo, and D. Rossi, "KISS: Stochastic Packet Inspection Classifer for UDP Traffic," *IEEE/ACM Transactions on Networking*, vol. 18, pp. 1505–1515, Oct. 2010.

[5] T. T. T. Nguyen and G. Armitage, "A Survey of Techniques for Internet Traffic Classification using Machine Learning," *IEEE Communications Surveys & Tutorials*, vol. 10, no. 4, pp. 56–76, 2008.

[6] T. T. T. Nguyen, G. Armitage, P. Branch, and S. Zander, "Timely and Continuous Machine-Learning-Based Classification for Interactive IP Traffic," *IEEE/ACM Transactions on Networking*, vol. 20, no. 6, pp. 1880–1894, 2012.

[7] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1." RFC 2616 (Draft Standard), June 1999. Updated by RFCs 2817, 5785, 6266, 6585.

[8] R. Kohavi, J. R. Quinlan, *Decision-tree Discovery*, ch. 16.1.3, pp. 267–276. Oxford University Press, 2002.

[9] N. Williams, S. Zander, G. Armitage, "A Preliminary Performance Comparison of Five Machine Learning Algorithms for Practical IP Traffic Flow Classification," *SIGCOMM Computer Communication Review*, vol. 36, pp. 5–16, October 2006.

[10] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA Data Mining Software: An Update," *SIGKDD Explorations*, vol. 11, no. 1, pp. 10–18, 2009.

[11] S. Zander and G. Armitage, "Design of DIFFUSE v0.4 - DIstributed Firewall and Flow-shaper Using Statistical Evidence," Tech. Rep. 110704A, Centre for Advanced Internet Architectures, Swinburne University of Technology, 2011.

[12] T. T. T. Nguyen and G. Armitage, "Training on Multiple Sub-flows to Optimise the Use of Machine Learning Classifiers in Real-world IP Networks," in *31st IEEE Conference on Local Computer Networks (LCN)*, pp. 369–376, November 2006.

[13] R. Pang and V. Paxson, "A High-level Programming Environment for Packet Trace Anonymization and Transformation," in *SIGCOMM*, Aug. 2003.

[14] "Anonymised FTP traces (lbnl.anon-ftp.Jan10-19.2003)." http://ee.lbl.gov/anonymized-traces.html.

[15] D. Bonfiglio, M. Mellia, M. Meo, D. Rossi, and P. Tofanelli, "Revealing Skype Traffic: When Randomness Plays with You," in *SIGCOMM '07: Proc. 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, (New York, NY, USA: ACM), pp. 37–48, Aug. 2007.

[16] "Anonymised Skype traces (E2E voice only and voice+video calls)." http://tstat.tlc.polito.it/traces-skype.shtml.

[17] L. Stewart and P. Branch, "SONG: Quake 3 Network Traffic Trace Files," Tech. Rep. 060406F, Centre for Advanced Internet Architectures, Swinburne University of Technology, 2006.