Measuring the Accuracy of Open-Source Payload-Based Traffic Classifiers Using Popular Internet Applications

Shane Alcock
Department of Computer Science
University of Waikato
Hamilton, New Zealand
Email: salcock@waikato.ac.nz

Richard Nelson
Department of Computer Science
University of Waikato
Hamilton, New Zealand
Email: richardn@waikato.ac.nz

***Abstract*—Open-source payload-based traffic classifiers are frequently used as a source of ground truth in the traffic classification research field. However, there have been no comprehensive studies that provide evidence that the classifications produced by these software tools are sufficiently accurate for this purpose. In this paper, we present the results of an investigation into the accuracy of four open-source traffic classifiers (L7 Filter, nDPI, libprotoident and tstat) using packet traces captured while using a known selection of common Internet applications, including streaming video, Steam and World of Warcraft. Our results show that nDPI and libprotoident provide the highest accuracy among the evaluated traffic classifiers, whereas L7 Filter is unreliable and should not be used as a source of ground truth.**

I. Introduction

In the field of traffic classification, the biggest problem facing researchers is the validation of their results. There are no labelled public datasets that can be used for this purpose, although recent projects such as [1] represent a significant step forward in this area. Instead, the most common source of ground truth when evaluating or developing new traffic classification techniques are the results produced by other existing traffic classification software. Payload-based methods are preferred in these situations because they typically offer the greatest accuracy [2].

However, this approach is only suitable if the existing technique achieves a high level of accuracy. If the technique produces incorrect or unreliable results then those errors may also be replicated in the new approach, given that the aim of the validation is typically to match the existing classifications as much as possible. This is especially problematic in the case of traffic that is designated as Unknown by the original approach; no ground truth is available for that traffic so it cannot be validated against. For example, if a classifier used to provide ground truth reports 30% of traffic as Unknown, a new technique will only be able to validate itself against 70% of the available traffic. Even if the new technique achieves 95% accuracy against the ground truth data, it has still only proved that it can correctly classify 66.5% of the total traffic in that dataset.

As licenses for commercial traffic classification software are expensive, many researchers use open-source alternatives to provide ground truth and validate their own work. While open-source traffic classifiers can be easily downloaded and used for this purpose, there is little published evidence that the classifications produced are accurate and reliable. L7 Filter [3], in particular, frequently appears in recent traffic classification literature, either as a source of signatures that were used to develop the classification engine [4] or to provide ground truth during validation [5] [6] [7], despite earlier studies suggesting that L7 Filter has a high rate of misclassification [8] [9].

However, if L7 Filter is unsuitable for this purpose, it is pertinent to ask which of the open-source traffic classifiers available to researchers should be used instead. Aside from the previously cited works which consider L7 Filter alone, there is an obvious lack of published material that evaluates and validates existing traffic classifiers with a sufficient level of depth to allow researchers to make an informed decision. We suspect that in many cases L7 Filter is chosen because it was used in earlier studies, rather than because it is the best available option.

In this paper, we present the results of a study designed to investigate and document the classification accuracy of four prominent open-source payload-based traffic classifiers. L7 Filter [3], nDPI [10], libprotoident [11] and tstat [12] were evaluated by running each classifier against packet traces captured from a single Internet end-host while using a variety of popular Internet applications. By adopting this approach, we can compare the accuracy of each classifier on a per-application basis where the ground truth has already been established. The evaluation included twenty-eight different applications or application protocols, including Steam, YouTube, Facebook, BitTorrent, RTMP and World of Warcraft.

Our principal findings can be summarized as follows:

1) L7 Filter produces inadequate results when used to classify traffic from many popular Internet applications. L7 Filter achieved a satisfactory level of accuracy for only six of the twenty-eight applications observed in our test dataset. Eleven of the twenty-two failures were

978-1-4799-0539-3/13/$31.00 ©2013 IEEE

applications supported by L7 Filter, suggesting that many L7 Filter signatures are incorrect.

2) nDPI achieved a very high level of accuracy in our study but still failed to correctly classify five of the twenty-eight applications. Provided full packet payload is available and the processing requirements of a deep packet inspection approach are not an issue, nDPI is demonstrably the best of the traffic classifiers that we evaluated, given that it can also subclassify HTTP traffic.
3) libprotoident also achieved a high level of accuracy, especially given it employs a lightweight approach that only examines the first four bytes of payload for each packet. In situations where deep packet inspection is not feasible (e.g. full payload is not available or processing speed is important), our results show that libprotoident is a reliable and accurate alternative.

II. Background

L7 Filter [3] is an application-level classifier that was originally designed for use with Linux NetFilter to perform traffic shaping and accounting. L7 Filter compares packet payload against a series of pre-defined signatures (described using regular expressions) and identifies the application based on what signature, if any, is matched by the packet payload. L7 Filter includes signatures for many application protocols, including well-known applications such as HTTP, BitTorrent, DNS and World of Warcraft. The most recent release of signatures was in May 2009, so recently released applications are unlikely to be supported by L7 Filter.

Historically, the L7 Filter signatures have been popular within the traffic classification community. L7 Filter is an open source project that was publicly released in 2003, when it was becoming apparent that port-based classification techniques were unreliable [13] [14]. At that time, researchers requiring a free deep packet inspection (DPI) tool to provide ground truth data for testing and evaluating classification techniques found that L7 Filter was the only viable option. Despite the appearance of other open-source DPI libraries and tools in subsequent years, L7 Filter is still frequently cited in recent literature in the traffic classification field. These citations often occur in the context of validating the results in the presented research or establishing ground truth, e.g. [5], [6], [7].

nDPI [10] is a fork of the OpenDPI project, which was an open-source release of an early version of the commercial PACE traffic classification engine. Following the purchase of the company that developed PACE, OpenDPI was removed from the public domain but the original algorithm and payload signatures have been preserved within the nDPI fork. The nDPI developers have subsequently extended the ruleset to include more recent applications and encrypted protocols.

Like L7 Filter, nDPI uses deep packet inspection to search the packet payloads for recognized patterns based on hand-written signatures. nDPI signatures are expressed in the C programming language rather than regular expressions which enables complicated patterns to be expressed more easily using branching logic. It also allows state to be retained across multiple packets so nDPI is able to match request-response patterns such as a 'GET' request and an 'HTTP' response. In addition, nDPI is able to examine the packet payload size and compare the size against the value present in a length field in the application protocol header.

Libprotoident [11] is a traffic classifier that has been designed to be effective in situations where deep packet inspection methods are not viable, e.g. due to privacy concerns or because DPI is too resource-intensive. Libprotoident uses a technique called 'lightweight packet inspection' whereby only four bytes of packet payload are inspected rather than the entire payload contents. This approach also means that libprotoident can classify public packet trace datasets that it would not be possible to analyze using DPI-based software, such as the ISPDSL-I and Waikato VIII datasets [15] where the packet payload has been truncated.

Aside from the differences in the amount of payload required, the techniques employed by libprotoident are very similar to those used by nDPI. The libprotoident classification rules are also expressed as C code and use a combination of payload signatures, payload size and port numbers to determine the application protocol for a given traffic flow. To match request-response patterns, flow state must be maintained by the program that is using libprotoident.

Finally, tstat [12] is an open-source traffic analysis tool that includes an application-level classification component that is based on deep packet inspection. Unlike the other tools, classification is not the primary goal of tstat; the software is intended for broader analysis of Internet traffic and therefore supports the fewer application protocols compared to the other classifiers. Much like nDPI, payload sizes and flow state are also frequently used to classify applications that do not have obvious payload signatures. Tstat is becoming increasingly prominent in literature and has been used as a source of ground truth in several recent studies [16] [17] [18].

III. Related Work

There have been several surveys of the traffic classification field in recent years, such as [2] [19] [20]. These reviews discuss traffic classification research at a broad level, summarizing the strengths and weakness of different classification approaches rather than investigating the accuracy and reliability of specific implementations. Dainotti et al. [20] describe the problem of obtaining ground truth to evaluate traffic classifiers, particularly given the difficulty in sharing sensitive data (e.g. full packet payload traces). The authors also mention the need for classifier implementations to be compared and evaluated against diverse Internet traffic in situations where the ground truth is already known, which reinforces the motivation for this study.

There has been some evidence presented in earlier studies that suggest that L7 Filter is not an ideal source of ground truth for traffic classification applications. Firstly, Canini et al. [8] evaluated the accuracy of L7 Filter against the ground truth obtained using their own iterative heuristic approach. The authors noted that L7 Filter produced many false negatives

Classifier	Version	Released	Protocols
Libprotoident	2.0.6	Nov. 2012	250
nDPI	1.3.0	Apr. 2012	149
L7 Filter	2009.05.28	May 2009	114
Tstat	2.3	Apr. 2012	42

TABLE I
THE EVALUATED TRAFFIC CLASSIFIERS.

for all traffic classes except for Web browsing, but this result was not strongly emphasized in the paper. This work differs from ours in that their evaluation of L7 Filter only considered accuracy by application category, e.g. mail, web, P2P, whereas our results are reported for each individual application. Also, the dataset used in their evaluation is a packet capture from a research facility during normal operation and is therefore biased towards the applications that are used in that context. Our dataset was manually generated to include a wide variety of well-known Internet applications.

Dusi et al. [9] conducted a study to evaluate the accuracy of both port-based and deep packet inspection techniques when classifying traffic at the application layer . The ground truth for this study was provided by gt [1], a tool that determines the application for a traffic flow using a daemon that associates each flow with the process that created it. This study concluded that deep packet inspection was better than port-based analysis, but still prone to a notable amount of error. However, the only example of a DPI classifier examined by this study was L7 Filter and it can therefore be argued that the error was not because of a flaw in the deep packet inspection approach but rather because of flaws in the L7 Filter implementation.

IV. METHODOLOGY

To evaluate the accuracy and utility of the traffic classifications produced by the open-source classifiers, we designed a series of experiments that would test each classifier in situations where the ground truth was already known. We captured packet traces from a single Internet end-host while we used a variety of pre-selected popular Internet applications. Each trace contained traffic for a single application. In this experiment, the Internet end-host was an iMac desktop running OS X 10.6.8. As application protocols tend to be independent of the operating system, we believe there is no bias in choosing OS X over Windows or Linux.

The host was located on a private RFC 1918 network and could access the public Internet via a gateway NAT device, but could not be connected to from outside the private network. This meant that unsolicited traffic and backscatter from the public Internet would not be present in our packet traces, without affecting the behavior or usefulness of the applications running on the end-host.

The packet traces were captured using the tracesplit tool included with libtrace [21]. The packets were captured by using the pcapint format module to listen on the network interface for the host machine. BPF filters were employed in most instances to remove unrelated traffic from the capture. e.g. ports 53 and 139 were filtered out of most of the traces. Traffic destined for multicast or broadcast addresses were also discarded during capture. The remaining captured packets, including all packet payload, were written to disk using the pcap file format without modification, i.e. no anonymization or truncation was performed.

The captured traces were then analyzed separately using each of the tools under consideration. The version numbers, release dates and number of supported application protocols for each classifier are shown in Table I. The classifiers were not modified in any way before running the analysis. After performing an initial analysis, the results for each trace were carefully inspected for the presence of traffic flows that were not intended to be part of the experiment. If such traffic was found, we removed it from the trace using tracesplit and repeated the analysis.

To run L7 Filter against our packet traces, we used the L7 Filter module included with the Traffic Identification Engine (TIE) [22]. This allowed us to utilize the L7 Filter regular expressions to classify traffic in userspace using our pcap packet traces as input. For this study, we used the unmodified default L7 Filter configuration included with TIE, as we anticipate this is how L7 Filter is most likely to be used. The TIE module was updated with the most recent release of the protocol signatures from the Clear Foundation. We configured TIE to retain the first 10,000 bytes of payload for each observed IP flow, which the regular expressions would be matched against.

The output from each classifier was merged into a single output file by matching lines from the individual classifier results that described the same flow five tuple (source IP, source port, destination IP, destination port, transport). The merged output was then manually analyzed to determine how well each of the four classifiers had performed, using our pre-existing knowledge of the application traffic that should be present in each trace file.

A. Subclassified Traffic

One problem that we had to account for during our analysis was the differing level of detail in the classifications produced by each classifier. For example, a YouTube video flow is usually classified as ‘flashvideo’ by nDPI but libprotoident will only recognize the same flow as ’HTTP’. We use the term *subclassification* to refer to situations where a classification provides additional detail about the purpose of the traffic flow beyond just the application protocol being used to transfer the data. In the above example, nDPI has subclassified the YouTube flow to indicate that the HTTP flow is being used to transfer Flash video rather than conventional HTML content.

nDPI subclassifies HTTP traffic by examining fields inside the HTTP header, particularly the Content-Type field, and making inferences accordingly. The differing interpretations are problematic when we are considering the correctness of a classification. While, technically speaking, the classification produced by libprotoident is accurate as the HTTP protocol is being used to transfer the video, the classification provided by

Application	Bytes	Complete Flows
DNS	44 KB	231
Facebook (HTTPS)	3.2 MB	229
FTP Control	2 KB	4
FTP Data	102 MB	4
Gtalk (XMPPS)	51 KB	12
IMAPS	2.1 MB	5
iTunes Store	127 MB	240
News Video	105 MB	322
POP3S	137 KB	15
PPStream	155 MB	1892
RDP	1.5 MB	4
SecondLife	35 MB	3632
Skype TCP	343 KB	49
Skype UDP	26 MB	99
SMTP	130 KB	2
Spotify	14 MB	1
Spotify P2P	10 MB	12
SSH	1.4 MB	2
Steam	7.2 GB	65
Steam Friends	1.7 MB	96
TF2 Game	2.4 MB	4
TF2 Server	330 KB	1143
TVNZ (RTMP)	56 MB	210
uTorrent	206 MB	278
Web (HTTP)	17 MB	657
World of Warcraft	902 KB	16
XMPP	224 KB	3
YouTube	52 MB	88

TABLE II
THE TRACES THAT WERE CAPTURED FOR THIS STUDY

nDPI better reflects the underlying application that generated the traffic. In the case of libprotoident however, it is unfair to say that a classification of 'HTTP' in our example is incorrect because the library was designed to classify traffic in cases where the information that would disambiguate the situation (the HTTP header fields) is not available. In such cases, HTTP is the best possible classification regardless of the application.

Therefore for this study, we have decided to ensure the comparisons are fair by treating both classifications as correct. In the given example, nDPI was more specific than libprotoident but, as the underlying protocol was still HTTP, libprotoident's classification would also be marked as correct in our analysis. However, we acknowledge the importance of subclassification, especially for HTTP traffic, and consider it to be a significant advantage of deep packet inspection techniques.

V. DATA SET

The four traffic classifiers were evaluated using a dataset consisting of traffic traces captured while using 28 different applications or application protocols. Where possible, we captured traffic for each application into a separate trace file but there were some cases where it was either not possible or not sensible to separate the application traffic in that fashion, e.g. the IMAPS and SMTP traffic were included in the same packet trace because the traffic was produced during the same email application session. In these cases, the initial capture was subsequently manually split into separate traces for each application that we were evaluating.

In this section, we describe the applications that were included in our data set in more detail and how each application was used. This is intended to allow the reader to be confident that our data set is fair and representative of contemporary Internet usage patterns. Note that whenever a web browser was used, the browser was Google Chrome. The quantity of traffic captured for each application is shown in Table II.

The applications that we evaluated and our usage of them were as follows:

DNS: For DNS, tracesplit was configured to only capture traffic observed on port 53 (both TCP and UDP). We then used a web browser to access a variety of popular URLs, including facebook.com, google.com, youtube.com, cnn.com, slashdot.org and nzherald.co.nz. We also entered a URL that we were confident would not be present in a local DNS cache (abcdefg.com).

Facebook: We logged into a personal Facebook account over HTTPS, browsed the news feed, uploaded a photo and played a Facebook game (Howzat Cricket). We also messaged and 'poked' other Facebook users who were in our Friends list.

FTP: Using a web browser, we went to ftp.kernel.org and browsed the directory listing. We then downloaded the latest version of the Linux kernel. When evaluating FTP, we considered the control and the data connections separately, as identifying the control flows is much easier than identifying the data flows where most of the traffic occurs.

Gtalk: Using a web browser, we logged into a Google account and used Google Talk to chat to friends. The chat used the XMPPS protocol, i.e. XMPP encrypted using SSL. We also attempted a Google+ "hangout" using video but this was unsuccessful.

IMAPS: Using the built-in Mail application in Mac OS X, we logged into an existing mail account using SSL-encrypted IMAP. We received new mail, moved mail between folders associated with the account and saved a draft email.

iTunes Store: Using the iTunes app in Mac OS X, we browsed the iTunes store, watched a video on iTunes U, listened to an audio podcast and watched a video podcast. iTunes uses HTTP and HTTPS for all data transfers and QuickTime as the preferred format for the podcasts.

News Video: Using a web browser, we visited several news websites to browse news articles and watch news video clips. The sites visited were the BBC (textual HTTP only), NZ Herald (textual HTTP articles and RTMP for video) and Fox News (Flash video embedded within HTTP).

POP3S: Using the built-in Mail app in Mac OS X, we accessed an email account using SSL-encrypted POP3 and downloaded some new mail messages to the local machine.

PPStream: PPStream is a Chinese P2PTV service that we have often seen in traces of residential traffic. We used the standard PPStream client to watch a single video, then left the client running for sometime afterwards to ensure we observed any P2P network maintenance traffic.

RDP: We connected to a Windows 8 PC on the same local network using Microsoft Remote Desktop Connection for Mac. Once connected, we browsed the open windows on the remote desktop before closing the connection.

SecondLife: We briefly played Second Life with an existing avatar using Second Life Viewer 2.

Skype: We logged into an existing Skype account using the 6.0.0.2968 Skype client for Mac. Once logged in, we added a new contact and video called another Skype user. When evaluating this application, we treated the TCP and UDP flows as using separate application protocols as some classifiers were capable of accurately identifying the UDP flows but not the TCP flows.

SMTP: Using the built-in Mail app in Mac OS X, we logged into an existing mail account via IMAP and sent an email. SSL encryption was used for both the login and the sending of the mail.

Spotify: Spotify is a music streaming service linked to Facebook that had recently become available in New Zealand. Spotify uses two protocols for streaming music which we evaluated separately: a protocol for streaming from servers owned by Spotify and a P2P protocol for streaming music from other Spotify users. Using the Mac Spotify application, we searched the database of available music and streamed a small selection of songs.

SSH: Using the scp command line tool, we copied a file to a remote host on the local network. We also logged in to the same host using ssh and did a directory listing to check that the file had copied successfully.

Steam: Steam is the leading digital game distribution platform at present. Using the standard Steam client application for Mac, we logged into a new Steam account, updated the Steam client and downloaded the free game Team Fortress 2. In evaluating the Steam application, we considered the Steam Friends protocol (used to communicate with fellow gamers through the Steam client) separately as it is quite noticeably different to regular Steam traffic.

TF2: Team Fortress 2 is a popular multiplayer team-based first-person shooter game that is played via Steam. Using the Mac Steam client, we started the Team Fortress 2 game, joined a game server and spent several minutes playing the game with another player. Neither the server nor the other player were located on our local network. When evaluating the Team Fortress 2 application, we have considered the server browsing protocol separately from the protocol used to exchange data about a game in progress.

TVNZ: To test the classifier's ability to identify RTMP, we used a web browser to watch a video from the TVNZ on Demand site. TVNZ is one of two major free-to-air television providers in New Zealand and is a big contributor of RTMP traffic in New Zealand [1]. The TVNZ on Demand content was delivered using the Brightcove streaming platform.

uTorrent: Using the 1.6.5 version of the uTorrent client for Mac, we downloaded a Debian ISO. Use of the uTP protocol over UDP for data exchange was enabled (as this is the default option), so much of the traffic was over UDP rather than TCP.

Web: In addition to the more specific web services, such as News Video and Facebook, we also used a web browser to perform conventional web browsing with the aim of producing a trace containing primarily standard HTTP traffic. We limited the capture to TCP port 80 only and visited several popular websites including Wikipedia, TVTropes, ESPNCricinfo and Stuff (a New Zealand news site).

World of Warcraft: Using the Mac OS X client for World of Warcraft, we briefly logged into the WoW Starter edition and played the game for a short period of time. The captured trace included only gameplay-related traffic; it did not include the download of the client or any game updates.

XMPP: Using Adium (a popular chat client for Mac) we used an account created on the jabber.meta.net.nz server to exchange messages with another XMPP user. We also transferred a file to the other user. Unlike with GTalk, the XMPP protocol was unencrypted.

YouTube: Using a web browser, we went to youtube.com and watched a single video. We then logged into a YouTube account, uploaded a short video, and proceeded to watch the video we had just uploaded. The YouTube traffic all used HTTP or HTTPS (much like iTunes) and the videos were Flash media contained within HTTP.

VI. Results

The accuracy results for the evaluated traffic classifiers against each of the traces in our dataset are shown in Tables III and IV. The accuracy values were calculated by iterating over each classification produced by the classifier in question and comparing it against the known ground truth for that traffic flow. A classification was deemed to be correct if it either matched the ground truth or was a suitable subclassification (see Section IV-A for more details on subclassification). A classification was marked as incorrect if the classifier produced a positive answer but the answer was incorrect, e.g. the classifier reported RTP for an SSL flow, or if the classifier reported the traffic flow to be unknown.

We considered a classifier to be able to successfully identify an application if it correctly classified more than 99% of *both* the bytes and flows in our test trace file. We defined a classifier as partially successful if the classifier correctly identified at least 95% of *either* the bytes or flows for an application but did not achieve sufficient accuracy to meet the preceding condition. This is intended to recognize situations where the classifier was generally correct but failed in certain edge cases. All other results were regarded as a failure.

To summarize our results, we found that two of the four traffic classifiers performed well when analyzing our dataset. Libprotoident (LPI) failed to correctly classify three applications, whereas nDPI outright failed on five applications. While neither classifier achieved a perfect result, a combination of nDPI and libprotoident would have produced correct classifications for every application that we examined except for Spotify P2P. nDPI was also able to provide more detailed

[1] Netflix, Hulu and other similar services are not available in New Zealand.

Application	Classifier	Bytes (%)	Flows (%)
DNS	LPI	100.0	100.0
	nDPI	100.0	100.0
	L7 Filter	98.1	97.4
	Tstat	100.0	100.0
Facebook (HTTPS)	LPI	100.0	100.0
	nDPI	100.0	100.0
	L7 Filter	31.5	39.3
	Tstat	100.0	100.0
FTP Control	LPI	100.0	100.0
	nDPI	100.0	100.0
	L7 Filter	100.0	100.0
	Tstat	0.0	0.0
FTP Data	LPI	100.0	100.0
	nDPI	0.0	0.0
	L7 Filter	0.0	0.0
	Tstat	0.0	0.0
Gtalk	LPI	100.0	100.0
	nDPI	100.0	100.0
	L7 Filter	81.4	83.3
	Tstat	100.0	100.0
IMAPS	LPI	100.0	100.0
	nDPI	100.0	100.0
	L7 Filter	0.0	0.0
	Tstat	100.0	100.0
iTunes Store	LPI	100.0	100.0
	nDPI	100.0	100.0
	L7 Filter	99.8	96.3
	Tstat	100.0	100.0
News Video	LPI	100.0	100.0
	nDPI	100.0	100.0
	L7 Filter	92.4	73.3
	Tstat	100.0	100.0
POP3S	LPI	100.0	100.0
	nDPI	100.0	100.0
	L7 Filter	10.6	13.3
	Tstat	100.0	100.0
PPStream	LPI	100.0	100.0
	nDPI	100.0	100.0
	L7 Filter	0.0	0.0
	Tstat	100.0	100.0
RDP	LPI	100.0	100.0
	nDPI	100.0	100.0
	L7 Filter	0.0	0.0
	Tstat	0.0	0.0
SecondLife	LPI	100.0	100.0
	nDPI	100.0	100.0
	L7 Filter	61.4	99.0
	Tstat	62.6	99.5
Skype TCP	LPI	11.2	61.2
	nDPI	99.8	98.2
	L7 Filter	11.2	61.2
	Tstat	11.2	61.2
Skype UDP	LPI	99.9	81.8
	nDPI	100.0	100.0
	L7 Filter	100.0	100.0
	Tstat	0.0	0.0
SMTP	LPI	100.0	100.0
	nDPI	100.0	100.0
	L7 Filter	100.0	100.0
	Tstat	100.0	100.0
Spotify	LPI	100.0	100.0
	nDPI	0.0	0.0
	L7 Filter	0.0	0.0
	Tstat	0.0	0.0
Spotify P2P	LPI	0.0	0.0
	nDPI	0.0	0.0
	L7 Filter	0.0	0.0
	Tstat	0.0	0.0

TABLE III
CLASSIFIER ACCURACY FOR THE FIRST 17 EVALUATION TRACES

Application	Classifier	Bytes (%)	Flows (%)
SSH	LPI	100.0	100.0
	nDPI	100.0	100.0
	L7 Filter	100.0	100.0
	Tstat	100.0	100.0
Steam	LPI	100.0	100.0
	nDPI	100.0	100.0
	L7 Filter	0.0	40.0
	Tstat	0.0	44.6
Steam Friends	LPI	100.0	100.0
	nDPI	0.0	0.0
	L7 Filter	0.0	0.0
	Tstat	0.0	0.0
TF2 Game	LPI	100.0	100.0
	nDPI	100.0	100.0
	L7 Filter	0.0	0.0
	Tstat	0.0	0.0
TF2 Server	LPI	100.0	100.0
	nDPI	0.0	0.0
	L7 Filter	51.1	85.1
	Tstat	0.0	0.0
TVNZ (RTMP)	LPI	100.0	100.0
	nDPI	100.0	100.0
	L7 Filter	2.5	87.6
	Tstat	17.5	97.6
uTorrent	LPI	100.0	99.6
	nDPI	100.0	98.6
	L7 Filter	100.0	95.0
	Tstat	100.0	98.6
Web (HTTP)	LPI	100.0	100.0
	nDPI	100.0	97.0
	L7 Filter	33.0	59.4
	Tstat	99.8	96.6
World of Warcraft	LPI	100.0	100.0
	nDPI	100.0	100.0
	L7 Filter	0.7	68.8
	Tstat	0.9	81.3
XMPP	LPI	100.0	100.0
	nDPI	100.0	100.0
	L7 Filter	100.0	100.0
	Tstat	100.0	100.0
YouTube	LPI	100.0	100.0
	nDPI	100.0	100.0
	L7 Filter	95.2	44.3
	Tstat	100.0	100.0

TABLE IV
CLASSIFIER ACCURACY FOR THE REMAINING 11 EVALUATION TRACES

subclassifications for much of the HTTP traffic in our dataset without introducing any false positives.

By comparison, L7 Filter was by far the worst performing classifier with our dataset. Of the 28 evaluated applications, L7 Filter correctly classified only six, with partial success being achieved for a further five applications. In particular, L7 Filter was unable to consistently identify traffic that had been encrypted using SSL and was also unreliable when presented with conventional HTTP traffic, particularly smaller flows. Given the frequency of these types of traffic on the modern Internet, this was an unsatisfactory result for L7 Filter.

Tstat was also disappointing, correctly classifying just 12 of the 28 evaluated applications. With the notable exception of Skype, the failures were primarily due to a lack of support for the applications in our data set. At this stage, it appears that Tstat lags behind nDPI and libprotoident when considered solely as a traffic classifier.

In each instance where a classifier failed to correctly identify an application, we investigated the cause of the failure so that we would be able to explain how the error occurred and, in some cases, how it might be corrected.

The results of our investigations follow:

DNS: 2.6% of the DNS flows were classified as Unknown by L7 Filter because the regular expression used to match DNS did not account for query names that contain underscore characters, e.g. SRV records. All of the other classifiers had no such problems classifying the DNS traffic.

Facebook: The reason for L7 Filter's failure with Facebook, as well as all other SSL encrypted traffic, is due to the classifier incorrectly reporting a significant portion of the encrypted traffic as Skype. This is because the first four bytes of one of the SSL handshake packets are (in hex) `0x16 0x03 0x02 0x00`. However, a common method for matching Skype traffic is to detect packets where the third byte is 0x02, creating the possibility of an erroneous classification if SSL traffic is not definitively ruled out first. In the case of Facebook, the error causes L7 Filter to incorrectly classify 48% of flows and 66% of the observed bytes as Skype. In addition, 11% of the flows were classified as Unknown by L7 Filter; these flows were correctly classified as either HTTPS or SSL by the other tools.

FTP: Both nDPI and L7 Filter were able to correctly identify the control traffic, but were unable to use this information to recognize the subsequent data transfer. Tstat had no support for FTP at all and therefore fails for both control and data.

Gtalk: L7 Filter again misclassified HTTPS traffic as Skype due to the same problem that affected the Facebook classification.

IMAPS: L7 Filter reported all of the IMAPS traffic as Unknown, except for one flow which was classified as RTP (Real-time Transport Protocol). This is because that flow was using SSLv2, which begins with the byte 0x80: the same byte that RTP payload can begin with. However, RTP uses UDP rather than TCP so this should be easy to disambiguate.

iTunes Store: L7 Filter fails to correctly identify some of the smaller HTTP flows, resulting in 3% of flows being classified as Unknown.

News Video: L7 Filter was unable to correctly classify the RTMP flow used by the NZ Herald video. The Fox News video used Flash video encapsulated in HTTP and L7 Filter did successfully identify these flows as HTTP, but it also missed a number of standard HTTP web browsing flows. Overall, L7 Filter incorrectly classified 27% of the flows in the packet trace, mostly by reporting them as Unknown.

POP3S: L7 Filter incorrectly identified 86% of the POP3S flows as RTP due to the same error from the IMAPS results.

PPStream: L7 Filter does not have a set of regular expressions for matching PPStream traffic, so all of this traffic was reported as Unknown. The other three classifiers had no problems identifying the PPStream traffic.

RDP: The signature for RDP in L7 Filter appears to be outdated, as all of the RDP flows were classified as Unknown by L7 Filter. Tstat misclassified all of the RDP traffic as SSL.

SecondLife: Both L7 Filter and Tstat correctly identified the TCP flows as HTTP but classified all of the UDP traffic as Unknown. Although, UDP traffic only constituted a small proportion of the observed flows, these flows accounted for 38% of the total observed bytes in the packet trace.

Skype TCP: Only nDPI was able to correctly classify TCP Skype traffic, which accounted for 1.2% of the bytes in the Skype trace. The other tools were only able to successfully identify the HTTP traffic generated by the Skype client, but not the Skype protocol itself.

Skype UDP: Libprotoident reports the occasional Skype UDP flow as Unknown. The source of the misclassification appears to be due to the first observed packet not having expected value of 0x02 in the third byte of payload. Tstat was unable to correctly identify any of the UDP Skype traffic in our data set.

SMTP: All four tools correctly classified all of the SMTP traffic, although the use case was relatively simple compared to the variety of SMTP traffic that can be seen on the Internet (e.g., no spam, no SMTP errors and we were using a mail server that we were authorized to use).

Spotify: Only libprotoident had rules for identifying Spotify traffic, however it was only able to identify the streaming traffic from the Spotify servers. The P2P traffic was labeled as Unknown by libprotoident.

SSH: All tools correctly classified the SSH traffic.

Steam: Tstat and L7 Filter were unable to correctly classify the flows associated with downloading the Steam updates or the game itself, reporting them as Unknown. They were able to identify the HTTP traffic associated with the Steam client, which is why they were successful for 40% of the flows.

Steam Friends: Only libprotoident had rules for identifying the Steam Friends protocol, which contributed 7% of the flows observed in the original Steam trace. The other tools all reported this traffic as Unknown.

TF2 Game: L7 Filter and Tstat classified the game traffic as Unknown, whereas the other two tools identified the traffic as belonging to the Half-Life protocol. The game uses the Half-Life 2 engine so it makes sense that the network protocol is very similar to the protocol used by the parent game.

TF2 Server: Tstat and nDPI classify the server browsing traffic as Unknown, whereas Libprotoident recognizes the traffic as "Steam UDP". L7 Filter managed to classify most of the server browsing flows as belonging to Team Fortress 2, but a small proportion of flows (including the larger flows) were reported as Unknown.

TVNZ: The current release of L7 Filter does not have a rule for RTMP (although one has been committed to the source code repository) so L7 Filter classified 97% of the traffic in this trace as Unknown. Tstat also failed to correctly identify the RTMP flows in this trace.

uTorrent: All four tools were excellent at classifying unencrypted BitTorrent, including the uTP flows. Only a small number of single packet flows were missed by each classifier.

Web: L7 Filter classified 40% of the flows and 67% of the bytes in the Web trace as Unknown. In each case, both of

the endpoints were behaving conventionally, i.e. the client had sent a standard GET request and the server had replied with a standard HTTP response. One possible reason for the problem is that the regular expression for matching HTTP in L7 Filter attempts to match based on the presence of certain fields in the HTTP response header, such as Content-Type or Date, rather than simply matching based on the payload beginning with "HTTP" as the other tools do.

World of Warcraft: The World of Warcraft protocol was recently re-designed as part of a game expansion released in late 2010. The rulesets for L7 Filter that we were using had not been updated to account for this change, so L7 Filter classified the World of Warcraft traffic in our trace as Unknown. Tstat did not support World of Warcraft and was therefore unable to classify the traffic in the trace.

XMPP: All tools correctly classified the XMPP traffic.

YouTube: Again, L7 Filter was unable to correctly identify many of the HTTP flows in the YouTube trace, although it did successfully classify the flows used to stream the video content. Over half of the flows were reported as Unknown by L7 Filter but these flows only accounted for 5% of the observed traffic. L7 Filter also misclassified some of the account login traffic as Skype.

VII. Conclusion

In this paper, we have presented an evaluation of the accuracy of several payload-based open-source traffic classifiers. The evaluation was performed by running each of the classifiers against a series of packet traces, each containing traffic belonging to one of a variety of popular Internet applications. The contributions of this work are twofold. First, we have validated and confirmed earlier studies that had indicated that the classifications produced by L7 Filter are not a reliable source of ground truth. Unlike previous efforts, we have evaluated the effectiveness of L7 Filter on a per-application (rather than per-category) basis and have documented the causes of many of the incorrect classifications.

We have also demonstrated that the failures of L7 Filter are not indicative of payload-based techniques as a whole, as was inferred by Dusi et al. [9]. nDPI and libprotoident were successful at correctly classifying most (although admittedly not all) of the applications that we examined and only one of the evaluated applications could not be classified by both tools. nDPI's ability to subclassify HTTP traffic suggests that it is the best option, provided full-payload packet capture is available. If not, our results demonstrate that libprotoident offers excellent accuracy given the minimal amount of payload required and is therefore worth consideration as an alternative.

As a final point, we note that L7 Filter has been frequently used as a source of ground truth by the traffic classification research community for several years, but our results suggest that it should be abandoned in favor of more reliable and accurate classifiers. L7 Filter's protocol signatures have not been updated since 2009 and many of the signatures that are present are either wrong or flawed, particularly the signatures for identifying web and SSL-encrypted traffic. Therefore, we wish to emphasize that there is little reason to continue using L7 Filter for this purpose, especially when libraries such as libprotoident and nDPI are freely available to researchers and have been shown to be more accurate than L7 Filter.

References

[1] F. Gringoli, L. Salgarelli, M. Dusi, N. Cascarano, F. Risso, and K. Claffy, "Gt: Picking up the Truth from the Ground for Internet Traffic," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 5, pp. 13–18, 2009.

[2] S. Valenti, D. Rossi, A. Dainotti, A. Pescapè, A. Finamore, and M. Mellia, "Reviewing Traffic Classification," in *Data Traffic Monitoring and Analysis*. Springer, 2013, pp. 123–147.

[3] Clear Foundation, "l7-filter," http://l7-filter.clearfoundation.com/.

[4] X. Yan, B. Liang, T. Ban, S. Guo, and L. Wang, "TrafficS: a Behavior-based Network Traffic Classification Benchmark System with Traffic Sampling Functionality," in *Neural Information Processing*. Springer, 2012, pp. 100–107.

[5] S. Dong, D. Zhou, W. Zhou, W. Ding, and J. Gong, "Research on Network Traffic Identification Based on Improved BP Neural Network," *Appl. Math*, vol. 7, no. 1, pp. 389–398, 2013.

[6] V. Carela-Español, P. Barlet-Ros, A. Cabellos-Aparicio, and J. Solé-Pareta, "Analysis of the Impact of Sampling on NetFlow Traffic Classification," *Computer Networks*, vol. 55, no. 5, pp. 1083–1099, 2011.

[7] M. Grajzer, M. Koziuk, P. Szczechowiak, and A. Pescape, "A Multi-Classification Approach for the Detection and Identification of eHealth Applications," in *Computer Communications and Networks (ICCCN), 2012 21st International Conference on*. IEEE, 2012, pp. 1–6.

[8] M. Canini, W. Li, A. Moore, and R. Bolla, "GTVS: Boosting the Collection of Application Traffic Ground Truth," *Traffic Monitoring and Analysis*, pp. 54–63, 2009.

[9] M. Dusi, F. Gringoli, and L. Salgarelli, "Quantifying the Accuracy of the Ground Truth Associated with Internet Traffic Traces," *Computer Networks*, vol. 55, no. 5, pp. 1158–1167, 2011.

[10] ntop, "nDPI," http://www.ntop.org/products/ndpi/.

[11] WAND Network Research Group, "libprotoident," http://research.wand.net.nz/software/libprotoident.php.

[12] A. Finamore, M. Mellia, M. Meo, M. Munafo, and D. Rossi, "Experiences of Internet Traffic Monitoring with Tstat," *Network, IEEE*, vol. 25, no. 3, 2011.

[13] T. Karagiannis, A. Broido, N. Brownlee, K. Claffy, and M. Faloutsos, "Is P2P Dying or Just Hiding?" in *Global Telecommunications Conference, 2004. GLOBECOM'04. IEEE*, vol. 3, 2004, pp. 1532–1538.

[14] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, "BLINC: Multi-level Traffic Classification in the Dark," in *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '05, 2005, pp. 229–240.

[15] WAND Network Research Group, "WITS: Waikato Internet Traffic Storage," http://www.wand.net.nz/wits/index.php.

[16] D. Adami, C. Callegari, S. Giordano, M. Pagano, and T. Pepe, "Skype-Hunter: A Real-time System for the Detection and Classification of Skype Traffic," *International Journal of Communication Systems*, vol. 25, no. 3, pp. 386–403, 2012.

[17] L. Grimaudo, M. Mellia, and E. Baralis, "Hierarchical Learning for Fine Grained Internet Traffic Classification," in *Wireless Communications and Mobile Computing Conference (IWCMC), 2012 8th International*. IEEE, 2012, pp. 463–468.

[18] J. Garcia-Dorado, A. Finamore, M. Mellia, M. Meo, and M. Munafo, "Characterization of ISP Traffic: Trends, User Habits, and Access Technology Impact," *Network and Service Management, IEEE Transactions on*, vol. 9, no. 2, pp. 142–155, 2012.

[19] H. Kim, K. C. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee, "Internet Traffic Classification Demystified: Myths, Caveats, and the Best Practices," in *Proceedings of the 2008 ACM CoNEXT conference*. ACM, 2008, p. 11.

[20] A. Dainotti, A. Pescapè, and K. C. Claffy, "Issues and Future Directions in Traffic Classification," *Network, IEEE*, vol. 26, no. 1, pp. 35–40, 2012.

[21] S. Alcock, P. Lorier, and R. Nelson, "Libtrace: A Packet Capture and Analysis Library," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 2, 2012.

[22] A. Dainotti, W. de Donato, and A. Pescapé, "Tie: A Community-Oriented Traffic Classification Platform," *Traffic Monitoring and Analysis*, pp. 64–74, 2009.