Statistical Protocol IDentification with SPID: Preliminary Results

Erik Hjelmvik
Independent Network Forensics and Security Researcher
Gävle, Sweden
erik.hjelmvik@gmail.com

Wolfgang John
Chalmers Universtiy of Technology
Göteborg, Sweden
wolfgang.john@chalmers.se

***Abstract*— Identifying application layer protocols within network sessions is important when assigning Quality of Service (QoS) priorities as well as when conducting network security monitoring. This paper introduces a Statistical Protocol IDentification algorithm (SPID) utilizing various statistical flow and application layer data features. We have identified application layer protocols by comparing probability vectors created from observed network traffic to probability vectors of known protocols. Promising preliminary results are presented, showing average precision of 100% and recall of 92% for a small set of protocols within traffic traces from an access network. To further improve the results, a number of ongoing and future directions with SPID are discussed, such as optimization of the attribute meters and improving robustness against different network environments.**

I. Introduction

Today, there is an increasing need for reliable classification of network traffic according to application layer protocols. Traffic classification is required for operational purposes, including QoS and traffic shaping mechanisms, optimization of network design, and resource provisioning. Furthermore, understanding the type of traffic carried on networks facilitates detection of illicit traffic, such as network attacks and related security violations. Modern firewalls, NATs and IPSs need to be able to reliably identify network protocols in order to implement fine-grained and secure access policies. Besides the apparent interest of operators and researchers to understand trends and changes in network usage, there have been a number of political and legal discussions about Internet usage (e.g. RIAA vs PirateBay), which further amplifies the importance of accurate traffic classification methods.

Currently, there are roughly four approaches to classify network traffic according to application protocols [1]. Traditionally, traffic was classified with sufficient precision by simply looking at TCP/UDP *port numbers*. With the advent of P2P file sharing systems and their legal implication due to copyright concerns, more and more applications started to use unpredictable dynamic port ranges or reused well-known ports of other applications, which yields poor results for port classification methods on modern network data [2], [3].

This development led to a wide use of *deep packet inspection* (DPI) for classification, which means inspection of packet payloads for known string patterns [4], [5]. DPI is currently the most reliable way to classify traffic, which explains its popularity in commercial tools. However, examination of (user) application layer data causes substantial legal and privacy concerns. Furthermore, DPI with static signatures is rather resource-expensive and does not work on encrypted traffic, which is becoming common as a reaction to legal threats.

As a result, the research community began to work on classification techniques independent from payload inspection. One such approach is to classify traffic based on *social behavior* of hosts by looking at their connection patterns [6]–[8]. While some of these methods are quite successful, they are only able to classify traffic in rough categories, such as mail, web or P2P traffic.

Another recent, payload independent approach is classification based on *statistical flow properties* such as duration, packet order and size, inter-arrival times, etc. [9], [10].

In this paper we introduce SPID, the Statistical Protocol IDentification algorithm, which was designed by Erik Hjelmvik [11]. The SPID framework is built to perform protocol identification based on simple statistical measurements of various protocol attributes. These attributes can be defined by all sorts of packet and flow data, ranging from traditional statistical flow features to application level data measurements, such as byte frequencies and offsets for common byte-values. In this sense SPID is a hybrid technique, utilizing efficient generic attributes, which can include *deep packet inspection* elements by treating them in the same way as *statistical flow properties*. A proof-of-concept (PoC) application for the SPID algorithm is available at SourceForge[1].

II. SPID Design Goals

The main goal of the SPID algorithm is to reliably identify which application layer protocol is being used in a network communication session in an easy and efficient fashion. SPID should not only be able to classify traffic into rough, coarse-grained traffic classes (such as P2P or web), but in fine-grained classes on a per-protocol basis, which would enable detailed QoS assignments and security assessment of network flows.

[1] http://sourceforge.net/projects/spid/

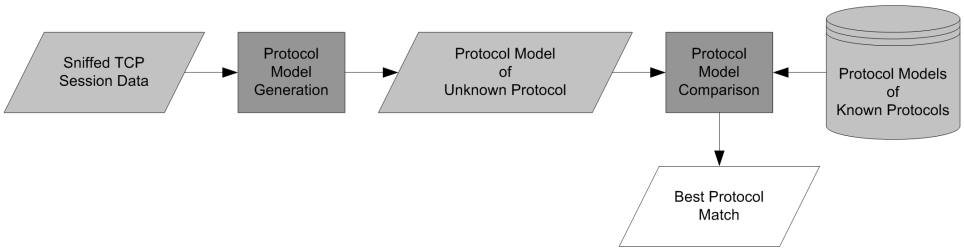

Fig. 1. Protocol identification data flow

Many application layer protocol identification schemes used today rely on signatures or patterns that usually occur in protocols, e.g. 'BitTorrent protocol', 'SSH-' or 'GET / HTTP/1.1'. A problem with looking for such static patterns is that the fingerprints need to be manually created, which means that network traffic and protocol specifications need to be studied and abstracted in order to create a reliable identification pattern. However, creation of application layer signature patterns can be automated, as shown by Park et al. [12].

Several protocols use obfuscation and encryption in order to prevent identification through static pattern-based signatures. Protocols that utilize such obfuscation techniques include the Message Stream Encryption (MSE) protocol (applied e.g. by BitTorrent) and Skype's TCP protocol. Manually creating signatures for proprietary protocols lacking documentation - such as Skype, Spotify's streaming protocol and botnet command-and-control (C&C) protocols - can be very troublesome.

An important design goal of SPID is therefore to replace the use of pattern matching techniques with entropy based comparisons of probability distributions. Doing so eliminates the need for manually extracting inherent properties of protocols, since the SPID algorithm has the ability to automatically deduce properties from training data. The training data used, however, needs to be pre-classified, which can be done through manual classification by experts or by active approaches, as in Szabo et al. [13]. A further goal of SPID is to allow protocol models to be updated easily as new training data becomes available, without having access to the previously used training data.

The required manual efforts for adding a new protocol are thereby shifted from detailed protocol analysis to assembling training data for that particular protocol. This is an important change since manual creation of static protocol patterns is a time consuming task, and new protocols continuously appear. Many new protocols are furthermore proprietary and undocumented binary protocols, which require advanced reverse engineering in order to manually generate protocol patterns.

The SPID algorithm does not require support for advanced pattern-matching techniques, such as regular expressions. By providing a generic XML based format to represent protocol model fingerprints, SPID is designed to be both platform and programming language independent.

Further operational key requirements for the algorithm are:

1) Small protocol database size
2) Low time complexity
3) Early identification of the protocol in a session
4) Reliable and accurate protocol identification

The motivation for requirements 1 and 2 are that it should be possible to run the SPID algorithm in real-time on an embedded network device with limited memory and processing capabilities. Requirement 3 should enable the use of the results from the SPID algorithm in a live traffic capturing environment in order to provide QoS to an active session in real-time, block illicit traffic or store related traffic for off-line analysis. An implicit goal is therefore that protocols should be identifiable based on the first few packets with application layer data.

III. Overview of the SPID framework

As illustrated in Fig. 1, SPID performs protocol identification by comparing the protocol model of an observed session to pre-calculated protocol models of known protocols.

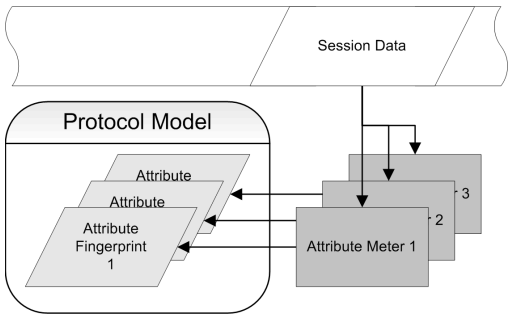

Fig. 2. Generation of protocol models

A. Protocol Models

Protocol models contain a set of *attribute fingerprints* (Fig. 2). Fingerprints are created through frequency analysis of various attributes, such as application layer data or flow features, and are represented as probability distributions. The PoC SPID application uses over 30 *attribute meters*[2], which are the functions that provide the distribution measurements for each specific attribute. An example of such an attribute meter is the basic ByteFrequencyMeter, which measures the frequency with which all of the possible 256 bytes occur in the application layer data. Other attribute meters perform much more advanced analysis of various properties in a session, such as measuring the frequency of various request-response

[2]http://spid.wiki.sourceforge.net/AttributeMeters

Index	0	...	80 ('P')	81 ('Q')	82 ('R')	83 ('S')	84 ('T')	85 ('U')	...	255
Counter vector	7689	...	1422	502	1001	1482	2644	961	...	3276
Probability vector	0.026	...	0.004	0.002	0.003	0.005	0.008	0.003	...	0.011

TABLE I
EXAMPLE OF AN ATTRIBUTE FINGERPRINT: BYTE FREQUENCY FOR HTTP

combinations (e.g. HTTP behavior, where a 'GET' request is followed by an 'HTTP' response or FTP behavior where a '220' message is replied to with a 'USER' command). The SPID algorithm also makes use of flow measurements (that do not require inspection of application layer data), such as packet sizes, packet inter-arrival times and packet order number- and direction combinations.

Attribute fingerprints are represented in the form of probability distributions. This means that the data for each fingerprint is represented by two arrays (vectors) of discrete bins: one array of counter bins and one of probability bins (Table I). Values of the counter vectors represent the number of times an observation (analyzed packet) has caused the associated attribute meter to trigger that particular index number in the vector. Probability vectors are normalized versions of the counter vectors, with all values in every probability vector summing up to 1.0. In this paper a vector length of 256 is used; an implementation of the SPID algorithm can, however, use any length for these vectors.

B. Generation of Protocol Models

For observed sessions, a protocol model is created upon session establishment (e.g. after the TCP three-way handshake), consisting of a set of attribute fingerprints. Every packet with application layer data belonging to a session is called an observation. Each such observation is then fed into the attribute meters, which provide measurements that are stored in the session's protocol model. Upon receiving such a measurement, the protocol model increments the fingerprint counters accordingly. For illustration, we assume an attribute fingerprint for the ByteFrequencyMeter from the first data packet observed in a HTTP session, i.e. a HTTP GET command. The counters would be incremented to

- 3 for the counter at index 84 (since there are three T's in 'GET / HTTP/ 1.1')
- 2 for counters at index 32, 47 and 49 (space, '/' and '1')
- 1 for counters at index 71, 69, 72, 80 and 46
- 0 for all other counters

All other attribute fingerprints belonging to the same protocol model will also increase their counters based on the sets of indices that are returned from their respective attribute meter. Subsequent packets in the same session will cause the fingerprint counter values to further increment. However, since one design goal of SPID is to keep time complexity low, we want to show in future work that utilizing only the first few packets provides sufficient precision.

Protocol models for known protocols are generated from real network packet traces. These traces need to be preclassified, either manually or automatically [13], in order to be usable as training data for the SPID algorithm. The preclassified training data is converted to protocol model objects (one per protocol) by generating protocol models for each session and merging (i.e. adding) the fingerprints of the same protocol and attribute type.

The more sessions are merged together for each protocol, the more reliable the fingerprint will be. As a rule of thumb, we found that 10% of the fingerprints' vector lengths (i.e. approximately 25) turned out to be a rough measurement of the minimum number of training sessions needed to build a reliable protocol model.

C. Comparison of Protocol Models

Fingerprints of an observed session are compared to fingerprints of known protocol models by calculating the Kullback-Leibler (K-L) divergence [14] (also known as relative entropy) between the probability distributions of the observed session and each protocol model, ranging from 0 (identical distributions) to ∞. The K-L divergence is a value that represents how much extra information is needed to describe the values in the observed session by using a code, which is optimized for the known protocol model instead of using a code optimized for the session protocol model. The best match for an observed session is the attribute fingerprint which yields the smallest K-L divergence according to Equation 1. P_{attr} and $Q_{attr,prot}$ represent the probability vectors for a specific attribute of an observed session and of a known protocol model respectively.

$$D_{KL}(P_{attr}||Q_{attr,prot}) = \sum_i P_{attr}(i) * log_2 \frac{P_{attr}(i)}{Q_{attr,prot}(i)} \quad (1)$$

Protocol models of observed sessions are finally compared to protocol models of known protocols by calculating the K-L divergences of the models' attribute fingerprints. The best protocol match is the one with the smallest average K-L divergence of the underlying attribute fingerprints. A good approach is to assign a threshold value, where only K-L divergence average values below the threshold are considered matches. If none of the known protocol models match, the session is classified as 'unknown' in order to avoid false-positives for known models.

IV. PRELIMINARY RESULTS AND ANALYSIS

For the following evaluation, the SPID Algorithm PoC application version 0.3 was used. The SPID PoC application was set to only analyze the first 20 TCP packets in each session and used a K-L divergence threshold of 2.25, which proved to be a good value after a series of empirical tests. However, a thorough evaluation of the impact of the treshhold value is subject of future work.

Protocol	Training Sess.	Validation Sess.	TP	FN	FP	Precision %	Recall %	F-Measure %
BitTorrent	31	1245	1221	24	0	100.0	98.1	99.0
eDonkey	19	3535	2744	791	0	100.0	77.6	87.4
HTTP	101	1333	1293	40	0	100.0	97.0	98.5
SSL	73	30	26	4	0	100.0	86.7	92.9
SSH	43	2	2	0	0	100.0	100.0	100.0

TABLE II
VALIDATION RESULTS PER PROTOCOL

We define a *session* as bi-directional TCP flows[3] identified by the 5-tuple[4]. Furthermore, an observed TCP three-way handshake (i.e. a *SYN* or *SYN+ACK* packet) followed by at least one packet with application layer data is required to qualify as a session suitable for classification by SPID.

The SPID PoC application is designed to only identify the application layer protocol in sessions that satisfy the flow requirements described above. The SPID algorithm can, however, be used to identify protocols in any communication scheme where there is a notion of a session, i.e. a uni- or bi-directional flow. This implies that the SPID algorithm can also be used to identify protocols that are transported or tunneled within other protocols such as UDP, HTTP, NetBIOS, DCE RPC, ISO 8073 or even SSL. This generalized functionality is not yet included in the SPID PoC application.

The training data for the protocol models is built from a collection of manually classified TCP sessions from private sources as well as public sources, such as DEFCON 10 CCTF[5], Honeynet.org[6], OpenPacket.org[7] and pcapr[8].

The validation data is a subset of a capture file provided by Szabo et al. [13]. This trace was collected on an access link with capture length of 96 bytes, ie. 42 bytes of application layer data. Since the sessions are pre-classified per client-side application, and applications might use multiple protocols concurrently, additional port filtering had to be used to generate a validation trace consisting of five TCP application layer protocols only:

- ***BitTorrent***: Azureus sessions, excl. ports 80, 10000, 10010 (HTTP) and 27001 (version check protocol)
- ***eDonkey***: eMule sessions, excl. port 80 (HTTP)
- ***HTTP***: Internet Explorer sessions, excl. port 443 (SSL)
- ***SSL***: Internet Explorer sessions to port 443 (HTTP)
- ***SSH***: PuTTY and WinSCP sessions

A. Validation Results

The results of SPID, with protocol models build from training data for these five protocols, are summarized in Table II. Following [15], *Precision* (or accuracy), *Recall* (or hit-ratio), and the combined *F-Measure* are defined according to Equations 2 to 4, where TP, FN and FP stand for *True Positives*, *False Negatives* and *False Positives* respectively.

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

$$F - Measure = \frac{2 * Precision * Recall}{Precision + Recall} \tag{4}$$

The results show that the SPID algorithm has the ability to perform very good identification of HTTP and BitTorrent sessions, while performing less complete for eDonkey (78% recall). On average, SPID yields the following results for the five protocols analyzed:

- *Precision*: 100.0% (no false positives)
- *Recall*: 91.9% (few missed sessions)
- *F-Measure*: 95.6% (combined measure)

B. Analysis of the Results

The low recall for eDonkey is believed to be due to the limited number of training sessions (only 19) available for this protocol. A richer set of training data will likely provide better results. However, eDonkey is known to be difficult to identify due to the very limited deterministic application layer data. While many existing classifiers are prone to generate false positives for eDonkey [16], SPID produces no false positives on the validation data with a KL divergence threshold of 2.25.

Both BitTorrent and SSH are considered easy to identify using application layer data, since they both start with static protocol banner strings [4], [17]. HTTP on the other hand is a more loosely described protocol, allowing much more freedom in the implementation. The fact that all HTTP traffic in the validation data stemmed from Internet Explorer traffic might explain why SPID achieved such good results, since the used HTTP training data mainly stems from web browsers. We would expect slighly lower recall if HTTP traffic from other applications had been present in the validation data set, such as traffic from HTTP tunnelers or SOAP based Web services.

V. ONGOING AND FUTURE WORK

Even though the current proof-of-concept application shows that SPID can successfully identify network sessions of various protocols, there is still a lot of work to be done.

As a first step, additional protocol models will be created and tested in order to be able to identify most modern Internet traffic. Besides the required training data for new protocol models, existing models will be enhanced with diverse training data from different network locations. It is therefore planned

[3] A bi-directional flow consists of the data sent in both directions
[4] A set of: source IP and port, destination IP and port, and transport protocol
[5] http://cctf.shmoo.com/
[6] http://www.honeynet.org/scans/
[7] https://www.openpacket.org/
[8] http://www.pcapr.net/

to get in contact with as many interested parties as possible in order to accumulate a database with protocol models with enough natural variation. These groups can include academic institutions, network developers, private individuals or any other interested parties. Please feel free to contact the authors if you would like to contribute.

Another crucial step will be to develop an improved validation framework. Since publicly available, pre-classified data as used in this paper (provided by Szabo et al. [13]) is very rare, we plan to adopt a similar approach like Kim et al. [15]. We will pre-classify our own data using an updated DPI method as introduced by Karagiannis et al. [6] in order to get a reference point when evaluating the performance of SPID.

We then plan to empirically test the accuracy of different attribute meters compared to pre-classified reference data. It is desirable to keep the number of attribute meters low in order to reduce CPU and memory complexity, so we hope to obtain a reduced, optimized set of meters. To each attribute meter we will provide recommendations, such as application protocols or network environments where this specific meter turned out to be especially powerful.

After defining an optimal set of attribute meters, the robustness of this set is planned to be tested against impact of the K-L divergence threshold and effects of different network environments. Some attribute meters, such as those depending on packet payload, are expected to perform similar in different environments. However, meters on flow features like inter-arrival times might be less robust when applied on flows from backbone links with much higher line-speed compared to LAN links. Besides LAN and access link data, we will have the possibility to test SPID on traces collected on 10Gbit/s backbone links [18].

Protocol identification on backbone links will, however, require some adjustments to the current SPID application. As shown in [19], routing symmetry on highly aggregated links is rare, which means that bi-directional flow data can no longer be assumed. Furthermore, attribute meters disregarding packet payload will become more important, since payload inspection on backbone links is often prohibited due to privacy concerns and legal implications.

VI. Summary and Conclusions

In this paper we presented SPID, the Statistical Protocol IDentification algorithm. SPID is utilizing various statistical packet and flow features in order to identify application layer protocols by comparison of probability vectors to protocol models of known protocols.

Initial results have been obtained when identifying a small set of protocols within a pre-classified set of flows collected on an access link. These results are very promising, showing 100% average precision with a recall of 92%. However, a number of interesting and relevant future directions with this approach are discussed, such as optimization of the flow features used or testing the robustness of the algorithm against different network environments, ranging from LAN to backbone links.

We believe that SPID has the potential to become a simple and efficient classification algorithm, providing accurate and fine-grained identification of network flows on application-protocol level. This is important for operational purposes, such as network provisioning, assignment of Quality of Service (QoS) priorities and network security monitoring. Furthermore, current discussions about legal aspects of P2P file-sharing applications add additional value to accurate traffic classification methods such as SPID.

Acknowledgment

The authors want to thank Geza Szabo, Atanas Atanasov, Hyunchul Kim, Tomas Karagiannis and the other authors of [15] and [6] for sharing their code and datasets. Furthermore, Erik is grateful to Jörgen Eriksson for his support and wants to thank his wife Sara for continuous support and motivation.

Erik Hjelmvik was supported by the Swedish Internet Infrastructure Foundation (.SE). Wolfgang John was supported by SUNET, the Swedish University Network.

References

[1] M. Zhang, W. John, K. Claffy, and N. Brownlee, "State of the art in traffic classification: A research review," *PAM Student Workshop*, 2009.

[2] A. W. Moore and K. Papagiannaki, "Toward the accurate identification of network applications," *PAM*, 2005.

[3] A. Madhukar and C. Williamson, "A longitudinal study of p2p traffic classification," *MASCOTS*, 2006.

[4] S. Sen, O. Spatscheck, and D. Wang, "Accurate, scalable in-network identification of p2p traffic using application signatures," *WWW*, 2004.

[5] L7-filter, "Application layer packet classifier for linux," *http://l7-filter.sourceforge.net/*, 2009 (accessed 2009-04).

[6] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, "Blinc multilevel traffic classification in the dark," *SIGCOMM*, 2005.

[7] W. John and S. Tafvelin, "Heuristics to classify internet backbone traffic based on connection patterns," *ICOIN*, 2008.

[8] M. Iliofotou, H. Kim, M. Faloutsos, M. Mitzenmacher, P. Pappu, and G. Varghese, "Graph-based p2p traffic classifcation at the internet backbone," *IEEE Global Internet Symposium*, 2009.

[9] J. Erman, M. Arlitt, and A. Mahanti, "Traffic classification using clustering algorithms," *SIGCOMM*, 2006.

[10] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli, "Traffic classification through simple statistical fingerprinting," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 1, pp. 5–16, 2007.

[11] E. Hjelmvik, "The spid algorithm - statistical protocol identification," *Technical Report, http://www.iis.se/docs/The_SPID_Algorithm_-_Statistical_Protocol_IDentification.pdf*, 2008 (accessed 2009-04).

[12] B.-C. Park, Y. J. Win, M.-S. Kim, and J. W. Hong., "Towards automated application signature generation for traffic identification," *NOMS*, 2008.

[13] G. Szabo, D. Orincsay, S. Malomsoky, and I. Szabo, "On the validation of traffic classification algorithms," *PAM*, 2008.

[14] S. Kullback and R. A. Leibler, "On information and sufficiency," *Annals of Mathematical Statistics*, vol. 22, pp. 49–86, 1951.

[15] H. Kim, K. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee, "Internet traffic classification demystified: Myths, caveats, and the best practices," *ACM CoNEXT*, 2008.

[16] E. Bursztein, "Probabilistic identification for hard to classify protocol," in *WISTP*, 2008.

[17] Y. Zhang and V. Paxson, "Detecting backdoors," in *In Proceedings of the 9 th USENIX Security Symposium*, 2000.

[18] W. John and S. Tafvelin, "Experiences from passive internet traffic measurements," *Technical Report 2008-17, Chalmers University of Technology*, 2008.

[19] M. Dusi, W. John, and K. Claffy, "Observing routing asymmetry in internet traffic," *http://www.caida.org/research/traffic-analysis/asymmetry/*, 2009 (accessed 2009-04).