This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE INFOCOM 2010 proceedings

PortLoad: taking the best of two worlds in traffic classification

Giuseppe Aceto, Alberto Dainotti, Walter de Donato, Antonio Pescapé
University of Napoli "Federico II" (Italy)
{giuseppe.aceto, alberto, walter.dedonato, pescape}@unina.it

***Abstract*— Traffic classification approaches based on deep packet inspection (DPI) are considered very accurate, however, two major drawbacks are their invasiveness with respect to users privacy, and their significant computational cost. Both are a consequence of the amount of per-flow payload data - we refer to it as "deepness" - typically inspected by such algorithms. At the opposite side, the fastest and least data-eager traffic classification approach is based on transport-level ports, even though today it is mostly considered inaccurate. In this paper we propose a novel approach to traffic classification - named *PortLoad* - that takes the advantages of both worlds: the speed, simplicity and reduced invasiveness of port-based approaches, on a side, and the classification accuracy of DPI on the other one.**

I. Introduction

Traffic classification has attracted increasing research efforts in recent years. This happened because, lately, the traditional approach of relying on transport-level protocol ports has become largely unreliable [1], while *deep packet inspection (DPI)* approaches have several limitations [2]. The main reason for the decline of port-based approaches is that several widespread applications started using random ports or ports assigned to other applications. Indeed, even if contained in the transport-level header, the two bytes of the port field are typically set by the application. Besides this, network applications also populate the transport-level payload with the headers of their protocols mixed in different ways with the application-level data. The result is that recurring strings of bytes related to the application protocol can be found in the transport-level payload, which has been instead exploited by traffic classification approaches based on DPI. However, even if DPI approaches are able to reach high levels of accuracy, they suffer of problems related to (i) their computational cost, which makes their deployment difficult and expensive, and (ii) users privacy, because they often require access to the full payload of all the packets (containing user data). Both issues are not present in the port-based approach.

The novel idea proposed and developed in this paper is that the philosophy at the base of the port-based approach - the verification of only few bytes in the first packet of a flow - can still be successfully used for traffic classification by moving the verification where the application protocol inserts its headers. This is done by (i) exploiting results from DPI techniques, (ii) studying the amount of payload information actually relevant in successful DPI matches, and (iii) purposely designing a fast packet-classification algorithm. We call the proposed approach *PortLoad* and, by providing a first practical implementation and preliminary experimental results, we show that it preserves the benefits of the port-based approach, as speed of execution and reduced invasiveness (associated with privacy concerns), while being able to reach accuracy levels close to those of DPI techniques. The preliminary results over real traffic, indeed, show classification accuracy percentages, normalized with respect to L7-Filter (a DPI classifier), of 97% ca. in the case of byte accuracy (compared to ca. 24% achieved by traditional port-based classification). As for speed, we experienced a 97.5% reduction in mean classification time compared to L7-Filter (and timings only 2.8 times higher compared to the port-based).

In the next section we provide details and motivations at the basis of the *PortLoad* approach. An implementation is described in Section III, whereas an experimental comparison, over a real traffic trace, among *PortLoad* and respectively a DPI and a port-based classification approach is performed in Section IV. This is performed in terms of classification accuracy, speed, and resource usage. We conclude the paper in Section V.

II. *PortLoad*: the approach

In this work we propose a novel approach to traffic classification that merges two different views: (i) the exploitation of application-protocol headers that are peculiar to network applications, which are carried by transport-level payload, and (ii) the inspection of only the first packet of a flow by attempting to match the fields of transport-level headers against a list of *signatures*, as it actually happens in port-based classification. In port-based classification a single packet - the first one - of a flow is examined in the transport-header port field; this classification approach can be considered a sub-case of packet-classification, in which the values of specific fields of packet headers compose classification rules, typically used to assign each single packet to a *flow*: in case of port-based approaches, a simple rule is given by the 16-bit number matching a list of IANA (and commonly known) assigned port numbers. In DPI approaches, instead, a pattern-matching algorithm is commonly executed on the whole content of all packets. An experimental analysis that we conducted however helps us understanding how *deep* into each single packet, and into each flow analyzed, a typical payload inspection classifier needs to go to identify applications associated with the traffic under observation. This is of interest because it represents a

[0]This work has been developed in the framework of COST action IC0703

978-1-4244-6739-6/10/$26.00 ©2010 IEEE

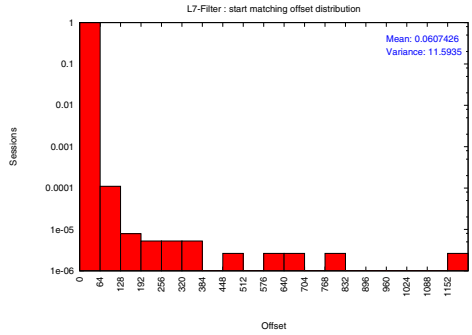

Fig. 1: Distribution of the offset of a matching string inside its first packet.

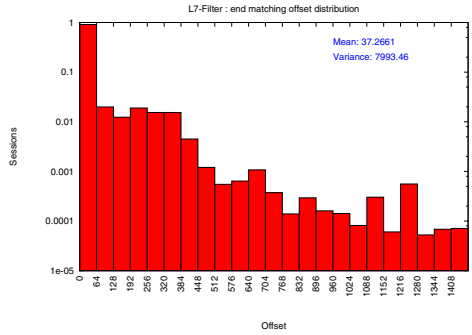

Fig. 2: Distribution of the offset of a matching string inside its last packet.

fundamental property, affecting both performance and privacy issues in traffic classification. We consider as reference L7-Filter, because it is the state of art of available payload inspection techniques and it was used in many scientific works for ground truth creation [3] [4]. We have been able to perform this investigation by using the TIE [5] classification plugin implementing L7-Filter techniques and adding it several debugging hooks for the collection of the following information: (i) the *number of attempts made for each session* (L7-Filter tries to match its regular expression rules against the stream of payload from a bidirectional flow (*biflow* in the following) every time a new packet is seen, by default up to the tenth packet); (ii) exactly *"where" the matches of regular expression rules happened*, in terms of (a) position of the packets inside the session, (b) position of the matching string inside the payload stream and (c) inside each packet. The experimental results here reported have been obtained by processing the trace described in Table IV. Observing the distribution of the number of attempts per session we found that 72% of the total attempts happen at the first packet. Moreover, considering that a successfully matching regular expression may span several packets, we computed the offset of its first character and last character from the beginning of the packets respectively containing them. The distributions of these two offsets, adopting a bin size of 32 bytes, are depicted in Figures 1 and 2 respectively. They show that almost all matching strings start (99.98%) and finish (90.77%) in the first 32 bytes of payload. These experimental observations (here summarized for space constraints) suggest that a totally different approach could be designed; an innovative approach trying to exploit the most significant amount of information generated by network applications that makes payload inspection approaches successfully work. The objectives when developing such new approach are (i) achieving a speed of classification that is comparable with port-based approaches, and (ii) designing a *privacy-friendly* technique. The three main guidelines in the design of the *PortLoad* approach are: (i) using only the first packet for each direction; (ii) using only the first few bytes of the considered packets; (iii) treating such bytes as fields to be inspected by packet-classification techniques. Such choices define an approach that fuses two opposite worlds: port-based and payload-based classification, which explains the name of *PortLoad*. It may be raised the objection that relying only on the first few bytes and on the first payload-carrying packet makes the circumvention of this classification approach easy, however protocol obfuscation techniques can be successful in general in circumventing all payload-inspection approaches based on string matching (circumventing port-based approaches is trivial).

III. *PortLoad*: THE IMPLEMENTATION

The proposed classification method associates biflows to one or more applications, each with a confidence value. The algorithm processes the first packet per direction of each biflow, and classifies it according to: the direction of the packet[1], the transport protocol, and the presence of application-specific bytes in fixed positions of the packet payload. This information together with a unique application identifier is collectively referred to as *signature* of named application (see Table I for an example). The algorithm is composed of two phases. The *rule matching phase* identifies multiple matching rules on each flow direction separately, and is based on elementary concepts of computational geometry that are typically used in fast packet-classification techniques. The *combining phase* performs evaluation and selection of the best combination of matching rules for the whole biflow (joining results related to both directions), and is based on a weight that is associated with each rule. According to the *PortLoad* classification approach, the algorithm for the rule matching phase, can be ascribed to the category of Packet Classification Algorithms (see [6]), and works by associating a *rule*, i.e. a set of values in fixed positions of the first packet, to an *action* to be taken for the processed packet (in this case, just the assignment of its biflow to a class, i.e. an *Application ID*). Contiguous positions of checked bits are grouped into *fields* and, in terms of computational geometry:

- each field defines a dimension of the D-dimensional space of processed packets;
- one packet is represented by a point in such a space;
- a rule defines an hyper-rectangle of the space;
- to classify a packet is equivalent to find the hyper-rectangles containing it.
- *rules collision* occur when hyper-rectangles of different rules overlap (a packet falling in the shared zone satisfies

[1] As the checked bytes fall in the L4 payload part of the packet, we actually refer to the *first payload-carrying packet*. The first packet of a bidirectional flow defines the upstream direction.

TABLE I: Example of a *PortLoad* signature for the *Shoutcast* MP3 streaming application (*App_ID* 34); the "⊙" symbol marks "don't care" value.

		direction		fields							
App_ID	TCP/UDP	UP/DW/BOTH	offset	1	2	3	4	5	6	7	8
34	UDP	BOTH	0	I	C	Y	\x20	⊙	⊙	⊙	\x20

all the colliding rules)

It is worth noting that, while in the common packet classification algorithms the bit *fields* are semantically linked with actual fields of L2, L3 and L4 headers and thus to the actions to be performed on packets, in our case this condition does not hold: the rules are related to different application protocols without *a priori* knowledge of a fields structure. However the underlying scheme of packet classification algorithms remains valid even if this link is relaxed. Therefore, by defining as separate *field* each group of 8 contiguous bits of the L4 payload, and considering at most D bytes of L4 payload, we characterize the rule matching algorithm as a search in the D-dimensional space for the hyper-rectangles (defined by the rule) that enclose the point representing the packet. More specifically, we propose a variant of the *Bitmap-Intersection classification scheme* presented in [7], with all fields of equal dimension (8 bits) and limiting the rule definition to either exact values or completely undetermined value ("don't care"). The following description is related to each flow direction. The simple idea behind the scheme is that if a point is enclosed in a hyper-rectangle, this must hold true in each single dimension for the projections of the point and the hyper-rectangle on it. In other words, a packet matches a rule if it independently satisfies the conditions on all the fields of the rule. The matching of a rule can then be made by checking one dimension at time, decomposing the main problem into simpler (and smaller) sub-problems, that can be also solved in parallel. So for each field index f (varying from 1 to the number of dimensions D, that is, the number of payload bytes considered), rules are grouped in sets defined on the basis of the value c they require for that field, as follows:

- *matching*, $M(c, f)$: rules checking for the value c in the field f;
- *compatibles*, $C(f)$: rules having no check for field f ("don't care");

For each possible packet, named $c_1, \ldots, c_D$ the values presented by the packet in the fields $1, \ldots, D$ respectively, the subset R of colliding rules is defined as:

$$R = \bigcap_{i=1,\ldots,D} M(c_i, i) \cup C(i)$$

A. The Matching Phase

The algorithm itself returns a set of successfully matching rules, and we exploit this to extract a subset of responses (see Sect. III-B). A fully parallelized implementation can be realized [7], but in order to adopt and test the presented classification scheme on general purpose architectures, we designed, implemented and tested an iterative version, described in the following. In order to find R, the iterative algorithm makes use of the following ancillary sets:

- *incompatibles* $I(c, f)$: rules checking in field f a value different from c;
- *terminated*, $T(f)$: rules that make their last check in field f (i.e. all fields after f are "don't care");

and the following temporary sequences of sets, iteratively populated one field at time during the matching process:

- *survivors*, S_f
- *found*, F_f

For each processed packet, the set S_0 of *survivors* is initialized equal to *the set of all available rules*, Ω, and the set F_0 of *found* rules is initialized as the empty set $\emptyset$. Then the sequence of the fields in the packet is scanned one at time, and each value is used to select the following subsets of rules, being c_f the value in the current field f:

$$F_f = F_{f-1} \cup (S_{f-1} \cap M(c_f, f) \cap T(f))$$

$$S_f = S_{f-1} \cap \widetilde{F_f} \cap \widetilde{I(c_f, f)}$$

where $\widetilde{(\cdot)}$ denotes the complement to Ω. It is evident that $M(c_f, f)$, $I(c_f, f)$ and $C(f)$ constitute a partition of the set of all rules Ω, so we have:

$$\widetilde{I(c_f, f)} = M(c_f, f) \cup C(f)$$

$$S_f = S_{f-1} \cap \widetilde{F_f} \cap (M(c_f, f) \cup C(f))$$

The result is that, at the iteration f, F_f will hold all the *terminated* rules that matched the packet values up to the field f, and S_f will hold all the rules that still could match for subsequent fields. The scanning is iterated until one of subsequent conditions occurs:

- the set of *survivors* rules S_f becomes empty (no rule would match anymore);
- the last field present in the packet is reached;
- the last field for rule definition (D) is reached;

After the last iteration, F_f will contain all and only the matching (colliding) rules, yielding to $R = F_f$, which will be processed in the *Combining Phase* (Section III-B). The iterative algorithm, given a set of rules, in the worst case will have computational complexity linear with the number of dimensions D, and varying with the number of rules according to the algorithm used for the intersection and union of the sets of rules. As in [7], in order to exploit bit parallelism for intersection and union operations, we represent the subsets defined above with bit vectors: said N the total number of rules, in a given order, the i-th bit of the bit vector will be 1 if the i-th rule belongs to the set, 0 otherwise. Said W the dimension in bits of the general registers of the underlying hardware architecture, each subset is implemented as an array of $n = ceil(N/W)$ elements, and each operation of intersection or union on the sets is implemented as n bitwise AND or bitwise OR, respectively. Furthermore, after having experimentally verified that rules starting with a large number of "don't care" fields may significantly delay the exhaustion

of the *survivors* set, we added an optimization (which should be tuned depending on the typical traffic breakdown present on the link under observation) to individually check such rules against the packet after the last iteration. This optimization is not needed in the case of a parallel implementation.

B. The Combining Phase

This phase is based on a comparison criterion among multiple matching rules for a single biflow with the purpose to assign to each candidate classification result (that is, the *app_id*, the unique identifier of the application) a confidence value. First, a *weight* is associated with each colliding rule. Such value is defined as the number of bytes composing the rule normalized to the maximum number of payload bytes considered by the algorithm (deepness of the inspection, see Section II): $weight_i = char_i/D$. The meaning of the weight derives from the fact that, in the (simplistic) hypothesis that the L4 payload is a sequence of independent uniformly distributed variables, the number of checked bytes is proportional to the self-information, or *surprisal*, related to the rule. Then, all the rules colliding on the biflow are grouped by the *app_id* they are associated with. For each *app_id* we do the following:

- the matching rules with the heaviest weights respectively in upstream and downstream are selected, and the sum of their weights is computed;
- because we experimentally verified that *reverse matches* are possible - that is, rules marked as UP can match on the downstream packet (and viceversa) - we also select the two heaviest *reverse matching* rules for each direction, but the sum of their weights is scaled by a penalty factor[2].
- the largest of the two sums calculated in the previous points is the *confidence value* assigned to the *app_id*.

At the end of this process, the algorithm output is an ordered list of responses in the form *(app_id, confidence)*.

IV. Experimental Evaluation

We compared *PortLoad* classification accuracy and performance (classification time and CPU/memory usage) against L7-Filter and a port-based approach (based on the CoralReef [8] signature set). We present the results obtained by processing the same trace used in Section II (Tab. IV) and with *PortLoad* signatures collected for 61 different applications, with a total of 554 signatures. The collection of the signatures has been done in three ways: (i) by manual adaptation from L7-Filter signatures, (ii) by payload inspection and manual selection, (iii) adapting some signatures used in [9] and listed at [10].

TABLE II: Overall Accuracy of *PortLoad* and Port-based.

Classifier	Accuracy on applications		Accuracy on categories of apps	
	sessions	bytes	sessions	bytes
PortLoad	74.24%	97.83%	73.88%	97.45%
Port-based	19.57%	25.12%	15.95%	23.89%

We first compare the classification results from *PortLoad* (with $D = 32$ bytes) and the port-based classifier against L7-Filter. To evaluate the (expected) loss in accuracy when moving from DPI to the *PortLoad* approach, we used L7-Filter as a ground truth tool and then measured the classification accuracy of *PortLoad*. *PortLoad* reported an overall classification accuracy of 74% and an overall byte-accuracy of about 97%, showing a very good accuracy on *heavy* flows. Table II also reports overall classification accuracy obtained by the port-based classifier, with (easily expectable) very low values. Fig. 3 represents the confusion matrix (with applications grouped into categories) of *PortLoad* against L7-Filter. The *high* colors on the main diagonal testify a good accuracy on most (categories of) applications, whereas few cells outside of it help to identify the application categories that are not well identified by *PortLoad*. In Fig. 4 we also summarized the classification results for the traffic classes associated with the largest byte-counts, where for each bar (corresponding to a class) it is illustrated the percentage of bytes on which *PortLoad* respectively agrees, disagrees, or returns *unknown*, with respect to L7-Filter. It is worth noting that: (i) a byte-accuracy of about 97% (when compared to a DPI approach) represents an astonishing result; (ii) about 40% of the traffic labeled as *unknown* by L7-Filter is instead classified by *PortLoad*. Further investigastions are required to understand if in some situations the signatures from *PortLoad* can overcome L7-Filter, and there are also issues related to the reliability of the ground truth used [11] that should be further investigated in the future.

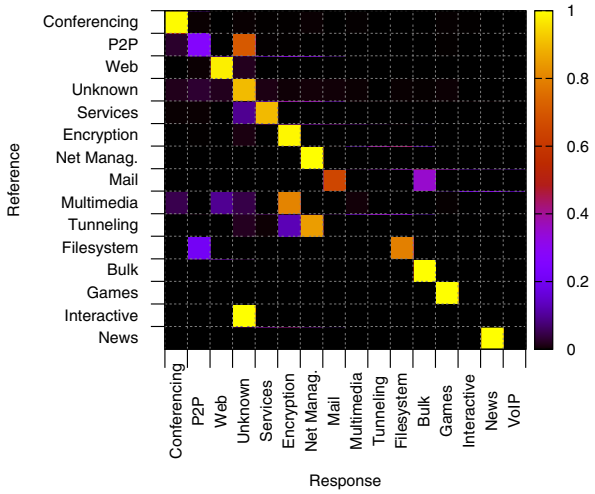

Fig. 3: Confusion Matrix of *PortLoad* classification (L7-Filter as ground truth).

As for the classification time, we used as a reference for minimum attainable classification times the values measured for the port-based classifier. During the tests, we verified that the classifiers were the only user processes running on the host. None of the system related processes were CPU intensive or I/O intensive. The Operating System was GNU/Linux running the kernel linux 2.6.27-8-generic. The overall results are summarized in Table III, where we report both the absolute values and a comparison with port-based performance. We found that *PortLoad* cuts mean classification time by 97.5% with respect to the DPI implemented by L7-Filter, reporting timings compa-

[2]In our tests, a value of 0.5 has been chosen.

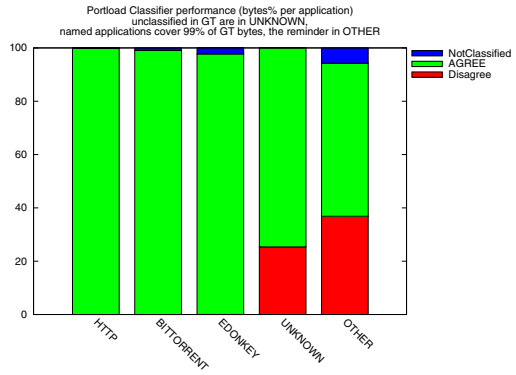

Fig. 4: Comparing *PortLoad* classification against *L7-Filter*.

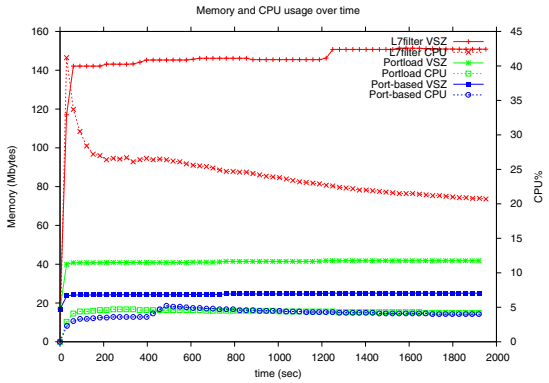

Fig. 5: Memory and CPU usage over time.

rable with the port-based classification performed by the port-based classifier (while still retaining much of the classification accuracy of payload inspection). These very promising results are consistent with the fact that the fixed string comparison done by *PortLoad* is much less computationally expensive than regular expression pattern matching (moreover, the higher variance is due to differences in length and complexity of the regular expressions in L7-Filter signatures), and that *PortLoad* signatures show small variability.

TABLE III: Classification Time Comparison.

Classifier	Mean Time (μsec)	Mean Time (*vs* Port-based)	Variance (μsec^2)
Port-based	2.48	1.0	0.88
PortLoad	6.99	2.8	11.15
L7-Filter	211.4	85.2	47057.88

As for the resources usage (see Fig. 5), regarding CPU[3], L7-Filter reaches the maximum in the first 60 seconds, we then see a smooth decrease to a steady value. The slow decay is probably due to the fact that the CPU% reported by `ps` is a moving average. Port-based and *PortLoad* show similar qualitative behavior, soon reaching their own steady level. Port-based reports a mean CPU usage of 3.7% (variance of 0.08), *PortLoad* 4.5% (variance of 0.11) and 25% for L7-Filter (variance 22.7). As for memory usage, the port-based classifier reports an average memory usage of 25193 KB (variance of 453300), 42266 KB for *PortLoad* (variance of 1258924) and 149217 KB for L7-Filter (variance 145426429). *PortLoad* presents a memory usage reduction of 72% with respect to L7-Filter. The much higher quantity of memory required by L7-Filter is due to the need to store more packets per session and more payload per packet.

[3]It is the CPU time used divided by the time the process has been running (cputime/realtime ratio), expressed as a percentage (`ps` man page).

V. Discussion and Conclusion

The preliminary results in this work show that *PortLoad* is a promising innovative classification technique taking the best features of two radically opposite approaches in traffic classification, while missing most of their drawbacks that often make such approaches inapplicable in modern networking scenarios. The values in the experimental evaluation section show that *PortLoad* may be the first traffic classifier with a combination of such high values of both accuracy and speed. For this reason future work will be targeted to the comparison of *PortLoad* against other traffic classification techniques (with a large and heterogeneous set of traffic traces). Moreover, another novel aspect of this approach is represented by a number of possible additional features that would require further study: (i) a user-adjustable trade-off between privacy (deepness D in the payload) and accuracy, (ii) the exploitation of multiple results with different confidence values in a multi-classifier approach, (iii) automated learning capabilities and signature generation. The latter could be done by employing algorithms presented in literature for similar purposes (e.g. automated common substring extraction) [12] [13].

TABLE IV: Details of the observed traffic trace.

Site	Date	Size	Pkts	biflows
Univ. Napoli	Oct 3rd 2009	59 GB	80M	1M

References

[1] T. Karagiannis, A. Broido, N. Brownlee, KC Claffy, and M. Faloutsos. Is p2p dying or just hiding? In *IEEE Globecom*, 2004.
[2] M. Morandi O. Baldini A. Monclus P. Risso, F. Baldi. Lightweight, payload-based traffic classification: An experimental evaluation. *ICC '08)*, May 2008.
[3] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli. Traffic classification through simple statistical fingerprinting. *ACM SIGCOMM CCR*, 37(1):7–16, January 2007.
[4] Zhu Li, Ruixi Yuan, and Xiaohong Guan. Accurate classification of the internet traffic based on the svm method. In *ICC*, June 2007.
[5] G. Aceto, A. Dainotti, W. de Donato, and A. Pescapé. Tie: A community-oriented traffic classification platform. In *TMA Workshop*, May 2009.
[6] P. Gupta and N. Mckeown. Algorithms for packet classification. *Network, IEEE*, 15(2):24–32, 2001.
[7] T. V. Lakshman and D. Stiliadis. High-speed policy-based packet forwarding using efficient multi-dimensional range matching. *SIGCOMM Comput. Commun. Rev.*, 28(4):203–214, 1998.
[8] *CoralReef*. http://www.caida.org/tools/measurement/coralreef/.
[9] T. Karagiannis, A. Broido, M. Faloutsos, and kc Claffy. Transport layer identification of p2p traffic. In *ACM IMC*, October 2004.
[10] T. Karagiannis. *Application-specific Payload Bit Strings*. http://www.cs.ucr.edu/~tkarag/papers/strings.txt, 2004.
[11] M. Pietrzyk, G. Urvoy-Keller, and J.L. Costeux. Revealing the unknown adsl traffic using statistical methods. In *TMA Workshop*, May 2009.
[12] P. Haffner, S. Sen, O. Spatscheck, and D. Wang. Acas: Automataed construction of applicatin signatures. In *SIGCOMM MineNet Workshop*, August 05.
[13] Byung-Chul Park, Young J. Won, Myung-Sup Kim, and James W. Hong. Towards automated application signature generation for traffic identification. In *IEEE NOMS*, April 2008.