On-line SVM traffic classification

Alice Este, Francesco Gringoli and Luca Salgarelli
Università degli Studi di Brescia,
via Branze, 38, 25123 Brescia, Italy
E-mail: <firstname.lastname>@ing.unibs.it

***Abstract*—A wide range of traffic classification approaches has been proposed in the last few years by the scientific community. However, the development of complete classification architectures that work directly in real-time on high capacity links is limited. In this paper we present the implementation of a machine-learning technique (SVM), one of the most accurate but most computationally expensive mechanisms, on the CoMo project infrastructure. We show the computational time required to process different traffic traces and the optimization steps we adopted to improve the performance of the system and achieve real-time classification on high-speed links.**

***Index Terms*—Computational time, On-line traffic classification, Machine-learning**

I. Introduction

Statistical traffic classification has been a very popular research topic in the past decade [1]. However, the deployment of these techniques on the Internet core is yet to come. One of the issues that still needs to be comprehensively analyzed and resolved is the scalability, in terms of computational complexity, of statistical classification algorithms when used to process traffic in real-time on high-speed links.

In this paper we describe the design, implementation and performance-optimization of a real-time traffic classification module deployed within the CoMo architecture [2], [3], an open source software for passive monitoring of network traffic. We selected a Support Vector Machine algorithm (SVM) since it has been proven to be one of the most accurate and, conversely, one of the most expensive traffic classification techniques [4], [5]. We focus on CoMo since it runs on inexpensive, widely available off-the-shelf Intel PC platforms. Furthermore, its modularity makes it relatively easy to extend it with new functionality and traffic analysis features.

We also characterize experimentally the performance of our CoMo classification module, focusing on the computational time required to process a set of different traces, and its memory occupancy. The results of the analysis show that the SVM module running under CoMo can process in real-time the traffic on links exceeding 1Gb/s running on commodity platforms without any HW-accelerating techniques. Since SVM is one of the most complex techniques for statistical traffic classification, this should be a proof that SW-based traffic classification can indeed scale to the capacity of modern access and even backbone links.

This work was supported in part by a grant from the Italian MIUR, under the PRIN project *IMPRESA*.

The paper is organized as follows: Section II presents related works. We describe the design and performance optimization of our CoMo SVM module in Section III. The traces we used in our performance characterization, together with the analysis of the experimental results, are presented in Section IV. Finally, Section V concludes the paper.

II. Related work

The tremendous growth of backbone link rates requires specialized approaches to traffic monitoring. Flow classification based on elementary rules built on the IP tuple (source and destination addresses, port numbers and transport protocol number) has been demonstrated to be feasible up 10Gb/s by exploiting Network Processor (NP) accelerators like the Octeon [6] and even at 100Gb/s on hardware designs based on FPGA like the Virtex-6 engine [7]. The implementation of a hybrid payload based-statistical algorithm for the recognition of Peer-to-Peer traffic has been recently demonstrated on an Intel IXP2400 NP [8]. Although these classifiers can indeed scale to very high capacities, they require specialized hardware, which besides increasing the cost of the monitoring nodes, they lead to implementations that are difficult to design, test and maintain.

Despite the processing power of NPs and the exceptional possibilities offered by FPGAs, some recent works bring general purpose platforms like off-the-shelf Intel-based workstation back to the traffic monitoring arena [9], [10]. These works show that new features like multi-core CPU and network interface cards with multiple TX/RX queues enable scenarios that just a few years ago were only achievable by costly custom hardware designs. Deri et al. in [9] demonstrate that targeted changes to the Linux kernel allow a commodity server to process and capture more than 4 Gbps per CPU. Smith et al. in [10] present an alternative high-performance network stack for the Windows kernel that is able to inject traffic exceeding 3Gbps on quad-core CPU. Finally, Ma et al. show in [11] that a Desktop platform equipped with a total of 8 cores can sustain a traffic exceeding 15Gpbs and classify packets given IP tuple based rules with no loss. The bottom line is that as new multi-core CPU architectures become available on the general market, it is not surprising to see applications monitoring 10Gbps links running on cheap off-the-shelf general purpose hardware. Contrary to our present paper, these works do not focus on statistical traffic classification, but only on the scalability of general traffic monitoring or traffic generation techniques.

978-1-4577-9538-2/11/$26.00 ©2011 IEEE

While the accuracy of classification systems based on machine-learning approaches has been widely studied, the evaluation of the required processing time in a complete classification system has been superficially investigated. Williams et al. [12] and Kim et al. [4] evaluated the computational requirements to run different machine-learning techniques, by adopting the WEKA tool implementation [13] of these algorithms. The evaluation of the processing time is performed off-line, after extracting from the traffic traces the set of relevant feature values.

In our previous work [14], we compared the computational complexity of the SVM-based classifier and a payload-based classifier. We identified and separated the sequence of operations each classifier performs in functional blocks, proposing a model to evaluate the most computational expensive parts and all the factors that influence the overall computational cost. Our experiments showed that in both classifiers the decision procedure is the most resource intensive stage.

In this paper we continue our analysis, showing by experimental results the limits of an SVM-based classifier when run on current commodity hardware. While there are a few approaches to modular SW-only classification architectures, such as [15] and [16], we focused our attention on CoMo mainly because it is currently used in several pan-european experiments. Therefore, it should be easy to reproduce our findings in other environments to confirm our results.

III. Classifier architecture

We integrated the SVM classifier as a new module inside the CoMo project [2] architecture. In this Section we review the main functionalities of CoMo and the features of the developed module.

A. CoMo architecture

CoMo [3] is a passive monitoring architecture released under the BSD License. It provides a software abstraction layer for different capture platforms and formats, supporting real time queries on the monitored traffic. Developers can add modules to extract and process relevant information from the packet stream, implementing plug-ins written in C.

In Figure 1 we show the architecture of the CoMo system, that is based on five different processes that control the data path through the system: packet capture, export, storage, query and supervisor. The plug-in modules are responsible

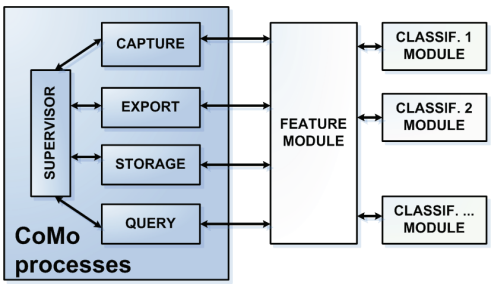

Fig. 1. Processes of the CoMo infrastructure and modules that perform classification of the traffic flows.

```
module "feature"
      description "Extract flow features"
      filter "udp or tcp"
      hashsize "100000"
      args "num_received_packets=4"
      args "flow_timeout=60" #(sec)
      args "interval=1" flush interval #(sec)
      args "classifiers=class_svm, class_bayes"
      args "classif_dir=path_to_classif_modules"
end
```

Fig. 2. Extract from CoMo configuration file to set the parameter values of the *feature* module.

for various data processing and trasformation. Each module implements a set of functions for exchanging data with each of the running processes. Users retrieve information about the monitored traffic by sending queries that are handled by the Query process.

B. Feature module

We implemented the *feature* module to aggregate packets into flows and to extract the features that the classification algorithms require. Figure 2 shows an extract from the CoMo configuration file to set the parameter values of the *feature* module.

We define the flow as the uni-directional ordered sequence of packets that share the same 5-tuple (source and destination IP addresses, port numbers and transport protocol). Flows expire after an inactive time interval of 60 seconds, in which no other packets with the same 5-tuple are seen. The flow definition we adopted is uni-directional; this choice works properly when packets are asymmetrically routed on the network nodes. Especially moving towards the core backbone links, the upstream frequently follows a different path than the downstream due to routing policy [17]. In the following we represent each flow by a set of features $\mathbf{x} = (s_1, s_2, s_3, s_4)$, where s_i represents the packet size of the $i-th$ observed packet. This choice was made according to [18] that proved that i) packet sizes carry enough information to detect the application layer protocol behind a network flow with high accuracy and ii) four packets are the optimal value. The EXPORT process invokes the classification algorithm when the fourth packet is received.

C. SVM Classification module

Classification libraries to be loaded are determined by the configuration parameter `classifiers` of the *feature* module (see Figure 2). The *classification* modules include the decision procedure by implementing the `classify()` function, invoked for each flow that transmits the minimum required number of packets.

1) SVM-based classification procedure: The SVM classification procedure considered in this paper is based on the Single-class Support Vector Machine algorithm, proposed by Schölkopf in [19], for which we evaluated the accuracy results in [5]. It requires a preliminary training phase to build a statistical model for each protocol class under examination. We used the pre-classification procedure adopted in [4] on a set of traffic traces with payload bytes we collected at our

University network, as we will explain in Section IV-A. The models calculated on this dataset were also applied to the other traces. Therefore, we adopted the classes as defined in [4]: *web*, *p2p*, *dns*, *mail/news*, *network operation*, *encrypted*, *chat* and *attack*; we excluded only the classes for which we cannot find enough flows in the training traces (*games*, *streaming* and *ftp*).

Each class is modeled as Gaussian–like test function that is used to determine the probability that a given observed flow $\mathbf{x}$ belongs to this class. The test value for the class j is then computed as follows:

$$f_j(\mathbf{x}) = \sum_{n=1}^{G} \alpha_n \, \mathcal{N}(\mathbf{x}|\mathbf{y}_{SVn}, \sigma) \tag{1}$$

where the number G of Gaussian terms $\mathcal{N}$, the weights α_n, the centers $\mathbf{y}_{SVn}$ (i.e., the *Support Vectors*) and the standard deviation σ, common to all terms, are determined during the training phase by analyzing a reasonable (in order of thousands) number of flows generated by this class. The number of extracted Support Vectors by the training procedure is comprised between 27 (*network operation* protocols) and 834 (*web* class)

2) Optimization: Gaussian function property: Equation 1 shows that the computation of the test function of each class requires to sum G multivariate Gaussian terms. The computation of each term involves an exponential function. However, the elaboration time required by the exponential function `exp()` of the C Standard Library, commonly used, can be reduced. The n-th addendum in Equation 1 needs to compute an exponential, that can be written as the product of one-dimensional exponential functions (since the covariance matrix is diagonal):

$$\exp\left(-\frac{||\mathbf{x}-\mathbf{y}_{SVn}||^2}{2\sigma^2}\right) = \prod_{i=1}^{d} \exp\left(-\frac{(x_i - y_{SVni})^2}{2\sigma^2}\right). \tag{2}$$

The one-dimensional function in Equation 2 can be pre-computed for all the possible values of $(x_i - y_{SVni})^2$. x_i and y_{SVn} range beetween 40 and 1500, corresponding to the minimum and maximum packet size. Therefore, for each class it is necessary to store a one-dimensional function of 1461 samples.

The use of this property of the multivariate Gaussian function allows to compute each addendum in Equation 1 as product of d pre-computed terms (corresponding to the number of features).

3) Optimization: Parallelization: The computation of the test function for a protocol class does not depend on the others. For this reason we parallelized the code by off-loading the evaluation of each test function in a separate thread. Since the SVM models we used in our experiments describe eight different traffic classes, we end running a total of eight worker threads. Unfortunately we faced a problem in the synchronization of the main code with the eight workers: using Mutex or even Futex to start the workers and acknowledge their outcome to the main thread revealed to slow down CoMo. We sorted out this issue by avoiding worker code to sleep during idle period and using atomic assignments to synchronize all the tasks without incurring in race-conditions. In this way we completely avoided using system calls at the price of loading eight cores at 100%, exploiting the available multi-core architecture. Though this can introduce a scalability problem, here the number of classes perfectly fits with the used hardware. We will investigate this issue in a future work.

Data set	Date	Type	Dur.	Traffic volume
UNIBS_A	Apr. 2, 2008	LAN	30 m	8.09 GB
UNIBS_B	Apr. 2, 2008	LAN	30 m	7.94 GB
AUCK._A	Jun. 8, 2001	LAN	5 h	2.27 GB
AUCK._B	Jun. 8, 2001	LAN	5 h	2.46 GB
MAWI_A	Dec 6, 2010	backbone	15 m	19.40 GB
MAWI_B	Dec 6, 2010	backbone	15 m	8.95 GB
CAIDA_A	Aug 14, 2002	backbone	5 m	12.06 GB
CAIDA_B	Aug 14, 2002	backbone	5 m	21.00 GB

TABLE I
DESCRIPTION OF THE DATA SETS. IN THE DATA SET NAME THE A/B INDICATES THE TRANSMISSION DIRECTION OF THE TRAFFIC.

IV. EXPERIMENTAL RESULTS

A. Data sets

The data set we used in our experiments is composed both of proprietary traces with payload bytes and publicly available anonymized traces. In fact, the classification procedure, based only on packet size values, can be applied even on anonymized traces. Table I summarizes the main properties of the traces, including LAN traffic to the Internet and backbone link packet collections. We keep separate the different transmission directions (we call directions A and B).

1) UNIBS data set: The packet traces composing the UNIBS data set were collected at the border router of our Faculty's network. The traffic traces were captured on the 100Mb/s link connecting the edge router to the Internet by running Tcpdump [20]. This data set includes the traffic of 30 minutes captured in the late morning, when the link load is higher, on April 2, 2008.

Since we have full monitor access to this router, we stored the first 250 bytes of every frame. Because we have the first payload bytes, we used the traffic, collected during another day on this same network point, to train the SVM models with reliable ground-truth. We achieved the ground-truth labels with the support of pattern-matching mechanisms applied on the payload, and in some cases with the addition of manual inspection.

2) AUCK. data set: The second set we consider, named AUCKLAND, is a trace collected at the University of Auckland in June 2001, available at [21]. The traces include only the header bytes, with a maximum amount of 64 bytes for each frame, while the application payload is fully removed. The header traces were captured with a GPS synchronized mechanism using a DAG3.2E card connected to a 100Mbps Ethernet hub interconnecting the University's firewall to their border router. This trace contains the traffic exchanged between the University and the Internet in 5 hours, with a lower

bit rate than the previous data set, as we can derive from Table I.

3) MAWI data set: The MAWI data set is a 15-minutes traffic trace from a trans-Pacific line (150Mbps link) in operation since 2006. This trace, indicated with Samplepoint-F, is publicly available on [22] in anonymized form, collected on December 6, 2010.

4) CAIDA data set: The CAIDA data set [23] contains anonymized traces collected at a commercial backbone link over a U.S. west coast oc48 peering link of a large ISP on August 14, 2002.

We used the MAWI and CAIDA traces to verify the applicability of our classifier to backbone links, where high transmission rates are common and the traffic sources can be more heterogeneous than in local networks.

B. Computational time and memory

In this Section we report about the measurement results we achieved by running the classifier implementation in the CoMo environment. We used a dual Xeon PC with a total of 24 cores at 2.6GHz running a 32-bit 2.6.35 Linux Kernel, with 48 GB of RAM.

Figure 3 shows the average time required to process 60 seconds of traffic for the four data sets. We obtained this value by dividing the overall processing time of the traces by the capture time. Backbone links, that transmit large amount of packets and carry high number of concurrent flows, need more time for their classification. The optimizations steps we applied reduced significantly the processing time, up to eight times in the CAIDA data set, making possible real-time classification even for that trace. The experimental results we reported in Figure prove that the developed classifier can work up to link bit rate of 600 Mbps (CAIDA direction B data set), requiring on average 21.5 seconds to process all the packets received in 60 seconds. Therefore, the remaining 38.5 seconds can be conveniently used to process traffic when the link capacity is higher.

We also evaluated the percentages of average CPU usage for the five CoMo processes. The only processes that shows significant CPU usage are CAPTURE and EXPORT (Figure 4). The former is in charge of the network sniffer and it extracts the flow identifier, according to the implementation of *feature* module. Its CPU usage increases when we introduce the optimizations as it has to perform the same operations in a smaller time. The latter process includes the time to manage the flow table, larger in backbone node and the classification time. EXPORT is the bottleneck of the system, in fact, Figure 4 shows that it uses 100% of the CPU, even introducing the first optimization. When we adopted the solution of dedicating a single thread for the computation of the test function of each of the eight classes, we see the average usage of the CPU for this process around 800-850%, since each thread runs on a different CPU.

We measured not only the computational time but also the memory requirements of the systems. As in the CPU analysis, we saw a significant impact only of CAPTURE and EXPORT

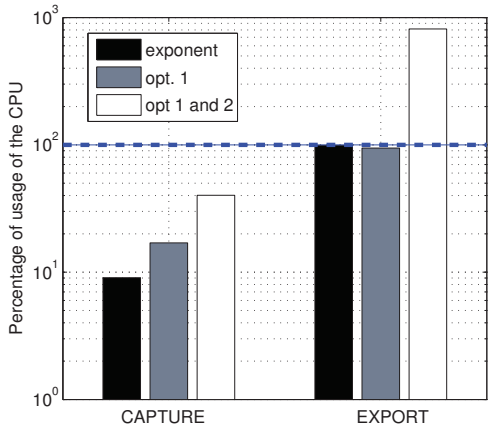

Fig. 4. CPU usage for the CAPTURE and EXPORT processes, when the UNIBS trace (direction A) is classified. Figures are similar for all traces. The last bar is valued at 814.4: CPU usage of the EXPORT process when both optimizations are applied is in fact related to the use of a single CPU. EXPORT uses up to 9 CPUs at the same time, when the eight classification threads are created.

processes. The memory requirements of the CAPTURE process, around 70-80 MB, are stable across the different traces. On the contrary, the EXPORT process needs more memory when backbone traces are processed. Figure 5 reports the values we measured to classify the four traces, when both the optimization steps are applied. The optimizations do not influence significantly the memory usage, in fact, we observed similar values also when they are not applied. This should not be a problem since the cost of memory is decreasing with a constant downward trend.

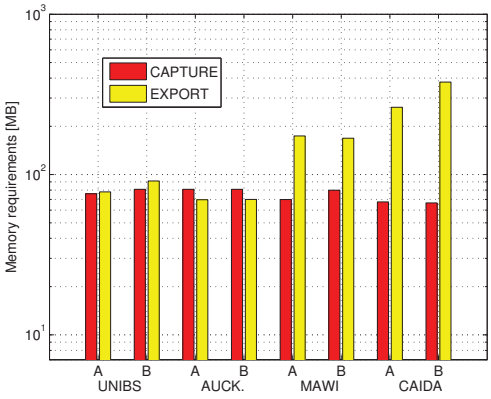

Fig. 5. Memory required by CAPTURE and EXPORT processes, when both the optimization steps are applied. The optimizations do not influence significantly the memory usage, in fact, we observed similar values also when they are not applied.

Finally, we show in Figure 6 the influence of the flow timeout value on the computational time and memory requirements. In all the previous experiments we use a timeout value of 60 seconds. In the right part of the graph, EXPORT process needs to keep memory of all flows for longer intervals, verifying if they are inactive before their removal from the table. We plan in the future to improve the performance of the system, checking the TCP signaling flags for an early removal

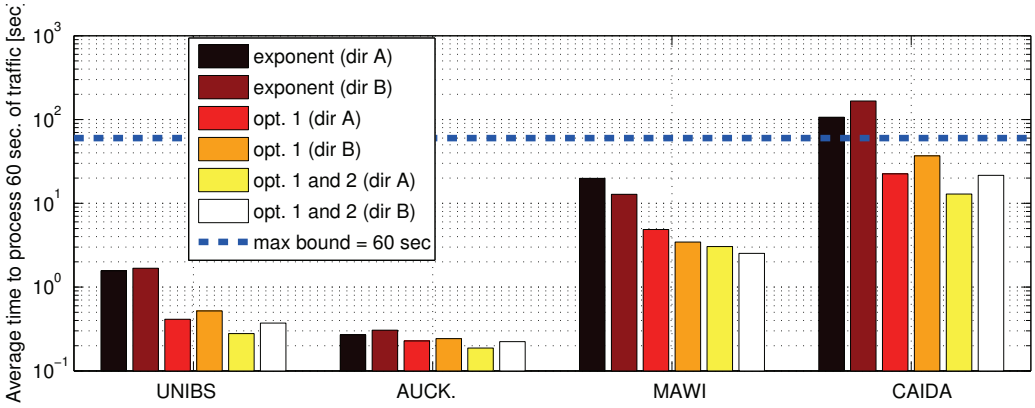

Fig. 3. Average time required to process 60 seconds of traffic for the four data sets. By adopting the code optimizations we achieved lower processing time on all the traces. With Opt. 1 we consider the Gaussian function property optimization, while with Opt. 2 we introduce the code parallelization.

of terminated flows.

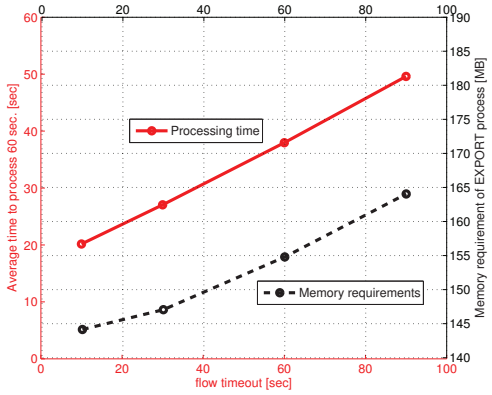

Fig. 6. Processing time and memory requirements for classifying the packets in the MAWI (direction B) trace, we measured by varying the flow timeout value.

V. Conclusions and Future Work

In this paper we presented the design, implementation and performance optimization of a machine-learning real-time traffic classifier based on the CoMo platform. Our experimental results show that such SW-only classifier, using only commodity HW, can process traffic on links up to 1Gb/s, even though we implemented one of the most resource-intensive machine-learning techniques at its base, Support Vector Machine.

The source code of two CoMo modules we developed (namely "feature" and "classification") will be available shortly on our website [24] with an Open Source license.

In the future we plan to verify whether our CoMo-based approach can be further optimized so as to make it scale beyond 1Gb/s links. The directions we are exploring are adding a DAG card to offload the capture process from the main CPUs, using processors with even more cores and possibly splitting the traffic from higher-capacity links so as to make it processable by multiple computers in parallel.

References

[1] T. Nguyen and G. Armitage. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys & Tutorials*, 10(4):56–76, 2008.

[2] G. Iannaccone. Fast prototyping of network data mining applications. In *Proceedings of the 2006 Passive and Active Measurement Conference (PAM 2006)*, Adelaide, Australia, Mar. 2006.

[3] The CoMo Project. http://como.sourceforge.net/.

[4] H. Kim, K. Claffy, M. Fomenkova, D. Barman, and M. Faloutsos. Internet Traffic Classification Demystified: The Myths, Caveats and Best Practices. In *Proceedings of CoNEXT'08*, Madrid, Dec. 2008.

[5] A. Este, F. Gringoli, and L. Salgarelli. Support Vector Machines for TCP Traffic Classification. *Elsevier Computer Networks*, 53(14):2476–2490, 2009.

[6] Y. Qi and L. Xu ad B. Yang. Packet Classification Algorithms: From Theory to Practice. In *Proceedings of the IEEE Infocom*, Rio de Janeiro, Brazil, Apr 2009.

[7] Y. Qi, J. Fong, W. Jiang, B. Xu, J. Li, and V. K. Prasanna. Multi-dimensional Packet Classification on FPGA: 100 Gbps and Beyond. In *Proceedings of the Intl. Conf. on Field-Programmable Technology (FPT'10)*, Beijing, China, Dec 2010.

[8] Z. Chen, B. Yang, Y. Chen, A. Abraham, C. Grosan, and L. Peng. Online hybrid traffic classifier for Peer-to-Peer systems based on network processors. *Elsevier Applied Soft Computing*, 9:685–694, 2009.

[9] F. Fusco and L. Deri. High speed network traffic analysis with commodity multi-core systems. In *Proceedings of the 10th annual conference on Internet measurement (IMC'10)*, Melbourne, Australia, Nov 2010.

[10] M. Smith and D. Loguinov. Enabling high-performance internet-wide measurements on windows. In *Proceeding of the 2010 Passive and Active Measurement Conference (PAM'10)*, pages 121–130, Zurich, Switzerland, Apr 2010.

[11] Y. Ma, S. Banerjee, S. Lu, and C. Estan. Leveraging Parallelism for Multi-dimensional Packet Classification on Software Routers. In *Proceedings of the 2010 International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'10)*, New York, USA, June 2010.

[12] N. Williams, S. Zander, and G. Armitage. A Preliminary Performance Comparison of Five Machine Learning Algorithms for Practical IP Traffic Flow Classification. *SIGCOMM Computer Communication Review*, 36(5):7–15, 2006.

[13] WEKA. http://www.cs.waikato.ac.nz/ml/weka.

[14] N. Cascarano, A. Este, F. Gringoli, F. Risso, and L. Salgarelli. An experimental evaluation of the computational cost of a dpi traffic classifier. In *Proceedings of the GLOBECOM Conference*, pages 1–8, Honolulu, Hawaii, USA, Dec. 2009.

[15] A. Dainotti, W. de Donato, and A. Pescapè. Tie: a community-oriented traffic classification platform. In *Proceedings of the Intl. Worksop on Traffic Monitoring and Analysis (TMA'09)*, Aachen, Germany, May 2009.

[16] DIFFUSE. http://caia.swin.edu.au/urp/diffuse.

[17] M. Crotti, F. Gringoli, and L. Salgarelli. Impact of Asymmetric Routing on Statistical Traffic Classification. In *Proceedings of the GLOBECOM 2009 Conference*, Honololou, Hawaii, USA, 2009.

[18] A. Este, F. Gringoli, and L. Salgarelli. On the Stability of the Information

Carried by Traffic Flow Features at the Packet Level. *ACM SIGCOMM Computer Communication Review*, 39(3):13–18, Jul. 2009.
[19] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, R.C. Williamson. Estimating the Support of a High-Dimensional Distribution. *Neural Computation*, 13:1443–1471, 2001.
[20] Tcpdump/Libpcap. http://www.tcpdump.org.
[21] Waikato Internet Traffic Storage (WITS). http://www.wand.net.nz/wits.
[22] MAWI Working Group Traffic Archive. Packet traces from wide backbone. http://mawi.wide.ad.jp/mawi/.
[23] The Cooperative Association for Internet Data Analysis (CAIDA). http://www.caida.org.
[24] UniBS Traffic analysis tools. http://www.ing.unibs.it/ntw/tools.