On the processing time for detection of Skype traffic

P.M. Santiago del Río, J. Ramos, J.L. García-Dorado, J. Aracil
High Performance Computing and Networking
Universidad Autónoma de Madrid
Madrid, Spain
Email: pedro.santiago,javier.ramos,jl.garcia,javier.aracil@uam.es

A. Cuadra-Sánchez, M. Cutanda-Rodríguez
Indra Sistemas, S.A.
Alcobendas, Madrid, Spain
Email: acuadra,mdcutanda@indra.es

***Abstract*—The last few years have witnessed VoIP applications gaining a tremendous popularity and Skype, in particular, is leading this continuous expansion. Unfortunately, Skype follows a closed source and proprietary design, and typically uses encryption mechanisms, making it very difficult to identify its presence from a traffic aggregate. Several algorithms and approaches have been proposed to perform such task with promising results in terms of accuracy. However, such approaches typically require significant computation resources and it is unlikely that they can be deployed in nowadays high-speed networks. In this light, this paper focuses on cutting the processing cost of algorithms to detect Skype traffic. We have conveniently tuned a previous well-validated algorithm and we have assessed its performance. To this end, we have used real traces from public repositories, from a Spanish 3G operator, and synthetic traces. Our results show that a single process can detect Skype traffic at 1 Gbps rates reading replayed real traces directly from a NIC. Even more, 3.7 Gbps are achieved reading from traces previously allocated in memory using a single process and 45 Gbps using 16 concurrent processes. This fact paves the way for 10 Gbps processing in commodity hardware.**

***Index Terms*—Skype; Traffic Classification; High-speed networks;**

I. Introduction

In recent years, the usage of IP telephony (VoIP) and particularly, of the Skype application, is becoming widespread. At the time of writing, it is estimated that Skype has over one half billion of users[1] (over 22 million users logged in simultaneously[2]) and it generated 185 million USD in the third quarter of 2009 (it is expected to reach 1 billion USD annual revenue in 2011)[1].

The analysis, characterization, classification and detection of Skype traffic is gaining considerable interest in the research community [1]–[4]. On the one hand, regulatory bodies are enforcing operators to intercept communications for security reasons (among which Skype calls). On the other hand, Skype's usage from mobile devices (such as smartphones, netbooks, etc) using 3G or GPRS mobile networks, is becoming very popular. For these reasons, operators are willing to detect Skype, either to provide differentiated quality of service or to restrict it or with billing/accounting purposes, depending on the contract. In any case, the detection of Skype traffic is becoming a very important issue.

As data transmission speeds have increased dramatically in recent years, the traffic classification applications are turning out to be a bottleneck for network monitoring. Note that current high-speed networks provide a bandwidth in the 1-10 Gbps range per link, due to the increasing popularity of new bandwidth-hungry applications like high-definition television, real-time data backup, gaming, to mention some of them. However, the performance evaluation of traffic classification algorithms in terms of processing time, and more specifically, Skype, have received relative little attention. We note that the most state of the art has focused on providing *accuracy* only, regardless of the processing power that is required, which may impair the practical applicability of the traffic classification algorithm in a real-world, high-speed environment.

This paper aims at filling this gap. Specifically, we seek for Skype detection algorithms that are both *accurate* and *fast*. Our analysis is focused in general-purpose servers, not specialized software. As it turns out, all Skype traffic classifiers found in the literature run in general-purpose servers. Actually, higher processing rates can be achieved with specialized hardware; however, the hardware solution is less flexible to incorporate changes to the algorithm and the cost is significantly higher. In our case, we trade off complexity of the detection algorithm versus responsiveness for real-time detection purposes, all in a general-purpose server. We should be able to answer some questions, such as "*Which is the limit rate, per CPU core, for detecting Skype traffic? Is a general-purpose server enough for traffic classification in a highly utilized 1 Gbps link? how about a 10 Gbps link?*". The answers of these questions are relevant for network operators with large backbone links, which seek for traffic classification at the minimum expense in monitoring equipment.

The authors in [1] proposed a method to detect Skype traffic based on two statistical techniques: on the one hand, based on the fact that traffic Skype is encrypted, they use Pearson's Chi-Square estimator to check if the packets' payload follows a uniform distribution; on the other hand, they use a Naïve-Bayes classifier to check if the packet length and the interarrival time fit with a hypothesized distribution of both magnitudes, which is inferred from the typical Skype codecs.

In this paper, we have borrowed some of these ideas, i.e., we have implemented an algorithm that follows a similar

[1]http://www.techcrunch.com/2009/10/21/skype-hits-521-million-users-and-185-million-in-quarterly-revenue accessed 16 March 2010

[2]http://skypejournal.com/2010/01/skype-dialtone-22-million-online.html accessed 16 March 2010

978-1-4244-9538-2/11/$26.00

approach to those statistical techniques, but conveniently tuned to satisfy the current high-speed network requirements. Then, we have evaluated its performance both in terms of accuracy and processing time.

We have evaluated our approach using real traces both from public repositories and from a Spanish ISP as well as synthetic traces. In order to evaluate the performance of our approach in terms of accuracy, i.e., false positives and negatives rates, we have used traces from Politecnico di Torino [5], which are labeled with ground truth, and synthetic traces that comprise P2P traffic. However, neither of these traces are representative of real traffic because they were gathered in controlled scenarios with few users or, directly, generated on purpose. Therefore, to evaluate the performance in terms of processing time, we have used a Spanish ISP trace which constitutes a good sample of the real traffic for a production traffic classifier.

First, our results show that our approach achieves similar results in terms of accuracy than previous work. More specifically, we obtain a percentage of false negatives of 6% in the worst case whereas the false positive rate is zero. Second, the performance of our proposal, in terms of throughput, shows that Skype traffic can be identified from a traffic aggregate of up to 1 Gbps with a single process reading replayed real traces directly from a NIC. Additionally, reading from traces previously allocated in memory gives a rate of 3.7 Gbps. It is worth remarking that these figures can be potentially scaled to the number of available cores using techniques as proposed in [6]. Specifically, in our general-purpose server with four 8-core CPUs, Skype traffic can be classified up at 45 Gbps rates reading from memory.

The rest of the paper is structured as follows: Section II outlines the existing work on the Skype application. Section III explains the design of our proposed Skype detection technique, *Skypeness*, whereas Section IV provides a performance evaluation in terms of throughput. Finally, Section V concludes this work with a brief summary of the main findings obtained and future work.

II. Related Work

Traditional services and protocols (such as FTP, web-browsing, SMTP, etc) are not difficult to detect by simple matching to well-known ports. However, such techniques are not enough to detect Skype traffic, which is a proprietary, obfuscated and encrypted protocol that uses per-session random ports. Therefore, not even access to the packet payload is granted and, consequently, well-known Deep Packet Inspection (DPI) [7] approaches are not longer valid. Because of this, the research community has proposed novel approaches based on the use of statistical traffic characteristics (or intrinsic traffic characteristics) and further applying, typically, Machine Learning (ML) techniques [8] to classify.

The authors in [9], [10] use ML techniques (such as AdaBoost, classification tree C4.5, SVM, etc) to design a simple classifier based on rules, starting from a large traffic trace. This classifier uses simple discriminants such as flow duration, bit rate, packet rate, protocol, as well mean, variance, minimum and maximum observed packet length. In addition, it is worth noting the existence of a U.S. patent [11], which claims the invention of a system and a method to build Skype traffic models and to further apply them to detect Skype traffic among other network traffic.

To the best of our knowledge, there is no state of the art in the performance evaluation of Skype detection from a computational point of view. The authors in [12]–[14] claim that detection of Skype flows is possible with the first 5 or 10 seconds of a given flow, with accuracy greater than 98%. Nevertheless, when a link (or a whole network) is monitored to detect Skype traffic, there are many different flows from other classes of traffic and the authors do not evaluate if their proposed technique can effectively discard non-Skype flows.

III. *Skypeness*

A. Detector Fundamentals

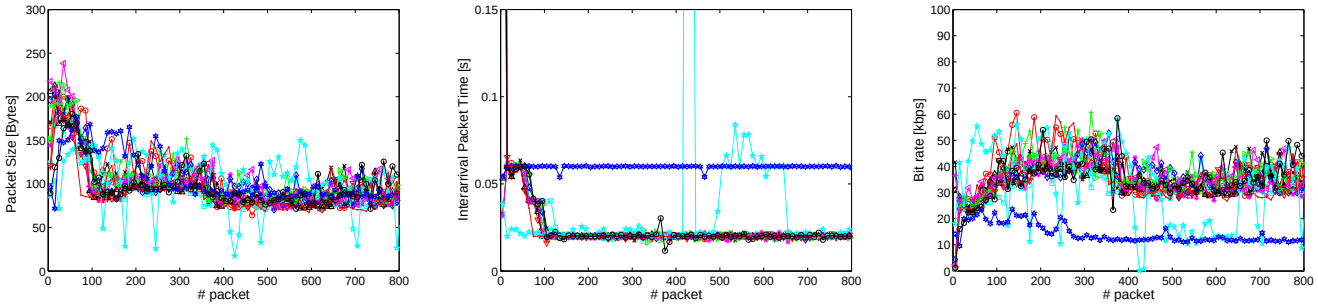
As stated the introduction, Skypeness test is based on the statistical techniques presented in [1]. We do not consider the Chi-Square estimator due to the high-performance requirements (Chi-Square requires inspection of the packet payload). Therefore, our detector uses the intrinsic characteristics of the Skype flows, namely: packet length, interarrival and bit rate. Fig. 1 shows the behavior of several audio UDP Skype flows in terms of these characteristics: packet length is delimited between 30 and 200 bytes (left), interarrival is nearly constant in multiple of 20 ms (middle) and bit rate is below 100 Kbps (right).

Flow information, such as timestamp (used to compute interrarivals) and size for each packet, is passed to the detector module. In order to smooth data, they are averaged in windows of 10 packets. It is worth noting that we only focus on audio UDP Skype flows with more than 30 packets (3 windows) because this is a validated trade-off threshold to detect Skype calls and ignore control flows. The detector computes the proportion of packet windows whose mean packet size, mean interarrival and mean bit rate are inside the valid intervals. If these proportions are greater than the given thresholds, the flow is classified as Skype. Table I shows the values for the intervals and the thresholds. The interval values have been chosen with an exhaustive study of Skype flows captured in several scenarios (wired and wireless connection, real and emulated networks conditions, etc). The thresholds values have been optimized using C4.5 trees, as in previous work [9], [10].

TCP Skype flows are not detected by Skypeness. However, it is worth noting that Skype uses UDP for voice transmissions as much as possible. TCP is used only when it is behind UDP-restricted firewall. Nevertheless, we plan to extend the classifier to TCP Skype voice calls and other classes of Skype traffic, such as video conferences, file transfers and chat.

B. Hardware and Software Architecture

Skypeness software runs over a general-purpose server based on 4 AMD Opteron 6128 processors working at 2 GHz. Each processor counts with 8 cores and the total memory is composed by 32x4 GB DDR3 memory boards working at



(a) Packet size (b) Interarrival (c) Bit rate

Fig. 1: Intrinsic Characteristics of a UDP Skype flow (audio conversation).

TABLE I: Intervals and threshold values used by Skypeness detector.

Characteristic	Interval	Threshold
Packet size [Bytes]	$[30, 200]$	0.75
Interarrival [ms]	$[i_{n-1} \pm 15]$	0.6
Bit rate [Kbps]	[0,150]	0.75

TABLE II: Skypeness NUMA nodes distance matrix.

NUMA Node	0	1	2	3	4	5	6	7
0	10	16	16	22	16	22	16	22
1	16	10	22	16	22	16	22	16
2	16	22	10	16	16	22	16	22
3	22	16	16	10	22	16	22	16
4	16	22	16	22	10	16	16	22
5	22	16	22	16	16	10	22	16
6	16	22	16	22	16	22	10	16
7	22	16	22	16	22	16	16	10

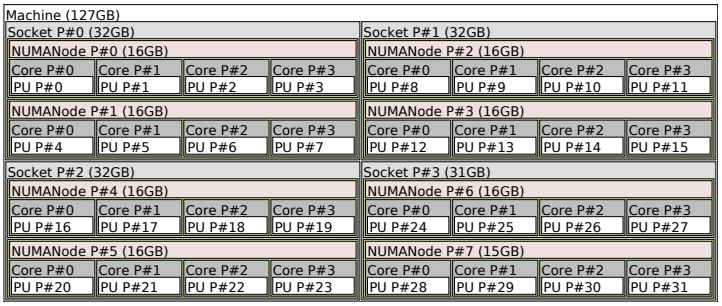


Fig. 2: NUMA architecture of *Skypeness*.

1333 MHz, on a standard Supermicro H8QG6 motherboard. To provide network connectivity one Intel 10 Gigabit CX4 Dual Port Server Adapter is used. This PCI-E 16x card uses an 82598EB controller that allows multiqueue transmission and reception up to 16 queues per interface and direction.

This server features a NUMA (Non Uniform Memory Access) architecture, whereby memory is split into several groups, one per CPU, giving raise to the so-called NUMA nodes (CPU+local memory). Clearly, better performance is achieved when the memory region of a process lies within the same NUMA node, such that the CPU executing the process only access local memory. However, the use of memory across different NUMA nodes increases access times and degrades computing performance. Fig. 2 shows the NUMA design for Skypeness hardware obtained using hwloc utility[3]. Following the rationale of better performance when memory locality is exploited, given a NUMA node, a distance vector to other NUMA nodes is defined. The lower the distance is, the higher the performance obtained accessing other NUMA node resources. Table II shows the NUMA distance matrix for Skypeness hardware obtained using numactl[4] utility.

On the software side, Skypeness runs over Ubuntu 10.04 Linux Server (64 bits) using 2.6.35 kernel. Skypeness is divided into three well distinguished modules. The first module is in charge of capturing and parsing incoming packets using a raw socket and mmap functions to map NIC receive queues at userspace level. Once a packet is processed, it is redirected to the second module responsible of creating and updating a list of flows or sessions. From now on, the paper will focus on flows rather than sessions for simplicity but all considerations could be applied as well to sessions. By flow, we mean a stream of IP packets sharing the 5-tuple (IP source and destination addresses, source and destination ports and protocol) and by session, we mean bidirectional flows. This module handles a hash-based flow table in memory to reduce access time to the bare minimum. All the memory used in the application is preallocated in a memory pool to reduce insertion/deletion time of a flow in the table.

Periodically, the active flows in the table are checked for expiration. To avoid checking all the active flows in the list, flows are sorted in decreasing order by last packet arrival time. Expiration time-slots happen every 15 seconds because it is the default expiration timeout specified by router manufacturers [15]. When a flow is marked as expired, it is deleted from the active flow list and redirected to the third module that is in charge of running Skype detection tests. If the flow passes all the tests, it is marked as a Skype flow and it is written into

[3]http://www.open-mpi.org/projects/hwloc/
[4]http://freshmeat.net/projects/numactl/

a file.

Note that this is a modular architecture that is based on a two-step treatment of flows, first flow extraction and then flow classification. In this case, flow classification is Skype versus non-Skype but it could be other. Actually, this software architecture gives flexibility and modularity to the detection tool, making possible the addition of other tests such as signature based DPI, etc.

IV. Performance Evaluation

A. Accuracy results

For the accuracy analysis, we have used three different traces. The first and second traces, named as Trace 1 and Trace 2 in the following, contain Skype traffic captured on the access link of Politecnico di Torino [5]. The set of users of such a network are typically students, faculty and administration staff. The measurement campaign duration was 96 hours in May/June 2006. Trace 1 only contains end-to-end Skype voice and video calls whereas Trace 2 only contains Skype out calls. Trace 1 contains ∼40M packets and Trace 2 contains ∼3M packets. The last trace used, named as Trace 3 in the following, is a synthetic trace captured in our laboratory at Universidad Autónoma de Madrid in August 2008. The trace contains ∼22M packets of P2P traffic from several applications, such as Emule and Bittorrent.

Table III shows the false positives/negatives rates in the traces described above. We only consider UDP flows with more than 30 packets, as stated in Section III-A. With traces 1 and 2 we only estimate the false negatives rate (these traces only contain Skype traffic). However, with Trace 3 we estimate the false positives rate (this trace does not contain Skype traffic). It can be observed that the false negative rate is below 1% in Trace 1 and is around 6% in Trace 2. On the other hand, Trace 3 shows a false negative rate equal to zero.

The obtained accuracy results are similar to the ones found in previous works which use trace 1 and 2: [14] shows a false negative rate near to 6% (in bytes) in the best case using only statistical classifiers (without inspecting packet payload). In [1], it is obtained a false negative rate greater than ours, when Naivë-Bayes classifier is used only.

B. Processing performance results

We have connected Skypeness to a server, which reproduces a pcap file using Tcpreplay[5]. This tool allows the transmission of pcap traces at variable rate. The transmission rate is varied during the tests (100, 250, 500, 750 and 1000 Mbps). We have found a limitation in the Tcpreplay throughput to 1 Gbps (i.e., we have not been able to send the pcap file faster than 1 Gbps in spite of using 10 Gbps NICs).

In this experimental setup, we have only set one reception queue and one traffic classifier instance running in the server. That is, we only use two cores: one for receiving packets and one for detecting Skype flows. Concerning NUMA affinity, we have set the CPU affinity of the reception queue to the NUMA node 1 and the CPU affinity of the Skype detector to the NUMA node 4 (the worst case in terms of distance).

For our performance analysis (processing point of view), we have used another trace, named as Trace 4 in the following. Trace 4 was captured from a 3G access network of a Spanish provider. The full trace contains traffic from residential households and small businesses. The trace contains ∼70M packets which correspond to ∼12M TCP/UDP flows captured during ∼18 hours in June, 2009.

The results are shown in Table IV. For each speed step, we can see the bit rate, the packet rate, the maximum number of flows expired (and consequently analyzed) in a second and the packet loss rate in the whole trace. It can be observed that there is no packet loss. It is worth noting that these results have been obtained using only two cores: one for receiving packets and storing them in memory and one for traffic classification. By using the technique proposed in [6], which assigns a reception queue per socket, we would be able to set up to 16 reception queues and 16 detection processes. In the following, we investigate if the use of this technique would allow performance gains of 16x, which would enable 10 Gbps Skype traffic classification in a general-purpose server.

To tackle this issue, some offline experiments were made. These experiments use a modified version of Skypeness software that obtains traffic from a local pcap trace instead of opening a socket for reception of frames. As it turns out, the theoretical read/write throughput of our DDR3 memory is 170.6 Gbps which is by far larger than the bandwidth of an Internet backbone link. To compute the hypothetical bandwidth that Skypeness can handle, the program was executed 10 times and execution times were obtained using Trace 4 as source. This methodology is repeated incrementing the number of parallel instances of Skypeness process and obtaining the corresponding execution times.

Taking into account the NUMA architecture described in section III-B the experiments have been designed such that data source and Skypeness software are located on different NUMA nodes and as far, in terms of NUMA distance, as possible. These conditions set out a worst case scenario. Fig. 3 and 4 show the execution time and the throughput versus the number of concurrent instances of Skypeness. It can be observed that the throughput of a single instance is 3.7 Gbps, scaling linearly up to a remarkable 45 Gbps classification speed using 16 instances. Note that slope is not 3.7 Gbps but lower. This is because every NUMA node serializes access to shared memory.

V. Conclusion and Future Work

In this paper, we have proposed an application for Skype traffic classification that works at 1 Gbps and 3.7 Gbps, reading from a NIC and from memory, respectively. In addition, we have assessed such application in our four 8-cores CPUs platform, providing a total throughput of 45 Gbps. From accuracy point of view, we have obtained a percentage of false negatives of 6% in the worst case whereas the false positive rate is zero, similar to related work.

[5]http://tcpreplay.synfin.net/

TABLE III: Accuracy Results.

Trace		Skype	Other	Class. Skype	Class. Other	FP (%)	FN (%)
Trace 1	Bytes	8381658970	0	8346887596	34771374	-	0.41
	Packets	39458562	0	39147589	310973	-	0.79
	Flows	1059	0	939	120	-	11.33
Trace 2	Bytes	231257652	0	217651943	182930	-	5.88
	Packets	3049148	0	39147589	2866218	-	6
	Flows	159	0	149	10	-	6.29
Trace 3	Bytes	0	1098935	0	1098935	0	-
	Packets	0	5312	0	5312	0	-
	Flows	0	52	0	52	0	-

TABLE IV: Performance results (per core) in packet, bit and flow rate.

Bit Rate [Mbps]	Packet Rate [Kpps]	Max. Flow Rate per second	Total Packet Loss Rate
100	30	26550	0
250	75	52800	0
500	150	90000	0
750	225	119000	0
1000	300	170000	0

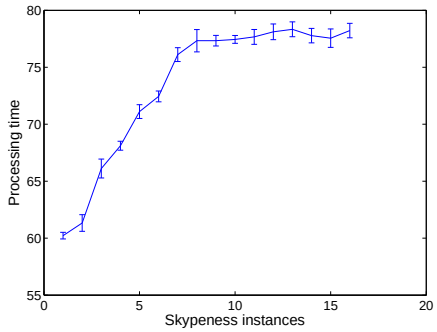


Fig. 3: Processing time obtained in offline processing.

These results show that identification of Skype traffic is feasible at the nowadays high-speed networks, typically ranging from 10 to 40 Gbps, using commodity hardware.

Given the difference, in terms of throughput, between reading from a NIC and from memory, we remark that traffic handling (both transmission and reception) with 10GbE NICs is the bottleneck of the process. In this light, we plan to address this issue tuning the current Intel driver and the Linux kernel.

We plan to apply the methodology of this paper to other classes of traffic, such as P2P, and to other detection techniques, such as DPI.

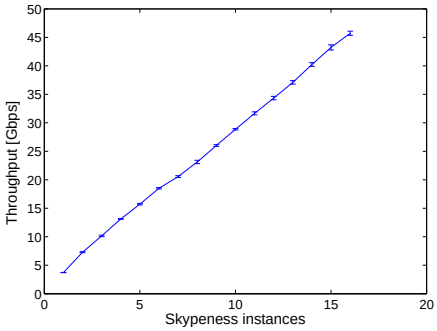


Fig. 4: Throughput obtained in offline processing.

REFERENCES

[1] D. Bonfiglio, M. Mellia, M. Meo, D. Rossi, and P. Tofanelli, "Revealing Skype traffic: when randomness plays with you," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 37–48, 2007.

[2] T. Hossfeld and A. Binzenhofer, "Analysis of Skype VoIP traffic in UMTS: End-to-end QoS and QoE measurements," *Comput. Netw.*, vol. 52, no. 3, pp. 650 – 666, 2008.

[3] L. De Cicco and S. Mascolo, "A mathematical model of the Skype VoIP congestion control algorithm," *IEEE Trans. Automat. Contr.*, vol. 55, no. 3, pp. 790 –795, 2010.

[4] D. Bonfiglio, M. Mellia, M. Meo, and D. Rossi, "Detailed analysis of Skype traffic," *IEEE Trans. Multimed.*, vol. 11, no. 1, pp. 117 –127, 2009.

[5] Telecommunication Networks Group Politecnico di Torino, "Skype traces: Traces from real internet traffic http://tstat.tlc.polito.it/tracce/Polito/2006/11_01_29_May_SKYPE_UDP_E2E.dump.anonim.gz," .

[6] S. Han, K. Jang, K. Park, and S. Moon, "PacketShader: a GPU-accelerated software router," *SIGCOMM Comput. Commun. Rev.*, vol. 40, pp. 195–206, 2010.

[7] N. Cascarano, L. Ciminiera, and F. Risso, "Optimizing deep packet inspection for high-speed traffic analysis," *J. Netw. Syst. Manage.*, vol. 19, no. 1, pp. 7–31, 2011.

[8] T.T.T. Nguyen and G. Armitage, "A survey of techniques for Internet traffic classification using machine learning," *IEEE Commun. Surv. Tutorials*, vol. 10, no. 4, pp. 56–76, 2008.

[9] D. Angevine and A.N. Zincir-Heywood, "A preliminary investigation of Skype traffic classification using a minimalist feature set," in *ARES*, 2008, pp. 1075–1079.

[10] R. Alshammari and A.N. Zincir-Heywood, "Machine learning based encrypted traffic classification: Identifying SSH and Skype," in *IEEE CISDA*, 2009, pp. 1 –8.

[11] S. Varadarajan, S. Gopalan, V.S. Basavaraja, and K. K. Rao, "System and method for Skype traffic detection. U.S. patent 20090116394," 2009.

[12] P. A. Branch, A. Heyde, and G. J. Armitage, "Rapid identification of Skype traffic flows," in *ACM NOSSDAV*, 2009, pp. 91–96.

[13] L.H. Do and P. Branch, "Real Time VoIP Traffic Classification," Tech. Rep. 090914A, Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, 2009.

[14] D. Adami, C. Callegari, S. Giordano, M. Pagano, and T. Pepe, "A real-time algorithm for Skype traffic detection and classification," in *NEW2AN and ruSMART*, 2009, pp. 168–179.

[15] Cisco Systems, "Introduction to Cisco IOS Netflow - a technical overview," *Cisco Technology White Paper*.