Service-Based Traffic Classification

M. Baldi, N. Cascarano, F. Risso, *Member, IEEE*

***Abstract*—This paper presents a novel approach in traffic classification that is based on the identification of the service that generates the traffic. This method is, in some sense, orthogonal to current approaches and it can be used as an efficient complement to existing methods to reduce computation and memory requirements. Experimental results on real traffic confirm that this method is extremely effective and may improve considerably the accuracy of traffic classification, while it is suitable to a large number of applications.**

***Index Terms*—About four key words or phrases in alphabetical order, separated by commas. For a list of suggested keywords, send a blank e-mail to keywords@ieee.org or visit** http://www.ieee.org/organizations/pubs/ani_prod/keywrd98.txt

I. Introduction

Traffic classification is one of the hottest topics in computer networks. On the one side, network managers want to know precisely the type of traffic transmitted over their networks to enforce various polices such as for quality of service (QoS), security, management, and more. On the other side, an increasing number of applications tend to hide their behavior (through encryption, tunneling, etc.) trying to avoid limitations imposed by such policies.

Traditionally, traffic classification relies on the port based method, which exploits transport layer information (source and destination TCP/UDP ports). However, this method has many limitations that make it quite imprecise and inefficient despite its extensive usage. Not all servers respect well-known ports conventions, malicious software can use well-known ports in order to let its traffic pass through port-based security restrictions, many peer-to-peer applications actively try to avoid classification using random ports, network tunnels can be instantiated using well known ports in order to avoid imposed traffic restrictions, IP payload encryption hides the port numbers.

An evolution of this approach relies on payload-based inspection that is used in most commercial devices and is declined in different flavors [4]. This technique shares some of the problems of port-based classification (encrypted protocols, tunneling) and is perceived as really expensive from the computational point of view. Other classification techniques that aim at identifying applications based on their behavior as inferred from observed traffic (statistic traffic analysis or heuristic analysis) are being studied, but are far from being ready for commercial deployment.

This paper presents a new classification technique that, in some respect, is orthogonal to the abovementioned mechanisms. Our approach, called *service-based classification*, exploits information about services previously discovered in the network in order to classify traffic flows. Main advantages of this method are robustness, accuracy, a limited use of processing power, reduced memory requirements, and the capability to use any classifier in the early stage of the classification (namely, the *service identification* phase).

This paper is organized as follows. Section II surveys the most common classification methods available in the literature. Section III describes the service-based classification idea, while some details about our implementation are given in Section IV. Section V presents an evaluation of this technique and conclusive remarks are presented in Section VI.

II. Related Work

Currently deployed network classification algorithms generally fall in one of two categories: payload based algorithms and behavioral algorithms. This section provides a brief overview of the state of the art in network traffic classification focusing on some of the most relevant algorithms in each category.

Payload-based classification is applied by most commercial solutions for various purposes ranging from statistics to security, because it provides the best trade-off between the classification accuracy and the coverage in terms of number of recognizable protocols. A possibly deep inspection of data transported within packets is used to identify the flow packets belonging to and the application generating it. In fact, by inspecting the headers of the higher layer protocols, possibly up to the application layer payload, it is possible to precisely identify the protocol being used by the application and

Manuscript received October 9, 2001. (Write the date on which you submitted your paper for review.) This work was supported in part by the U.S. Department of Commerce under Grant BS123456 (sponsor and financial support acknowledgment goes here). Paper titles should be written in uppercase and lowercase letters, not all uppercase. Avoid writing long formulas with subscripts in the title; short formulas that identify the elements are fine (e.g., "Nd–Fe–B"). Do not write "(Invited)" in the title. Full names of authors are preferred in the author field, but are not required. Put a space between authors' initials.

F. A. Author is with the National Institute of Standards and Technology, Boulder, CO 80305 USA (corresponding author to provide phone: 303-555-5555; fax: 303-555-5555; e-mail: author@ boulder.nist.gov).

S. B. Author, Jr., was with Rice University, Houston, TX 77005 USA. He is now with the Department of Physics, Colorado State University, Fort Collins, CO 80523 USA (e-mail: author@lamar.colostate.edu).

T. C. Author is with the Electrical Engineering Department, University of Colorado, Boulder, CO 80309 USA, on leave from the National Research Institute for Metals, Tsukuba, Japan (e-mail: author@nrim.go.jp).

possibly gather information on the type of traffic it generates. However, the correct identification of a protocol is not straightforward. One approach relies on searching for patterns or regular expressions that can uniquely identify each protocol; a database containing the description of each protocol is needed. Many payload based solutions have been proposed [2] [3], some coupled with an approach for describing network protocols in order to make classification code easy to reuse and update [5][6]: classification of additional protocols or new versions of existing protocols can be achieved by simply adding their description, without the necessity of any modification to the classification software itself.

Known problems of payload based classification algorithms are (*i*) high sensitivity to packet loss and TCP/IP fragmentation and segmentation issues, (*ii*) hard and time-consuming task of creating protocol signatures, that are crucial to the effectiveness of the solution, (*iii*) encryption and/or tunneling that hinders access to data contained into application layer headers and payloads, and (*iv*) significant requirements in terms of computational and memory resources that actually make traffic classification at high line rates difficult.

Due to the high computational requirements of deep packet inspection, payload based classification algorithms usually limit pattern searching to the initial packets of each flow. According to this method, named Packet Based – Flow State in [4], once the protocol transported by a flow has been recognized, the flow identifier (i.e., the 5-tuple including IP addresses, ports, and transport layer protocol) and the corresponding application-layer protocol are added to a data structure in memory, often called *session table*, that is maintained as long as the flow is active[1]. The main critic moved toward these methods is about the memory usage for maintaining flow state information; in case of large networks, the size of such per-flow state grows significantly and this might become an issue. Furthermore, additional memory is required because pattern matching usually relies on regular expressions, which are well-known for their memory consumption due to the necessity of maintaining graph-based structures representing Deterministic Finite Automata. On the other side, also processing requirements may be problematic due to regular expression matching and to session table management (lookup, insertion, deletion, etc.). These problems become even worse in the Message Based – Protocol State flavor [4] of the payload-based method (implemented in Binpac [6] and SML [7]), that needs to rebuild the entire application-layer message to enable the analysis of the entire data in order to achieve the precision required for security appliances. In this case, the amount of information to be maintained grows even more, as do processing requirements for session reconstruction and application-layer processing, although some smart method can be devised in order to decrease this complexity [18]. It is important to notice that [4] demonstrates that the simpler Packet Based – Flow State approach is in most practical cases sufficient for the vast majority of applications.

Another approach in traffic classification relies on behavioral techniques, whose main assumption is that each application is characterized by some specific behavior. Applications can then be identified by just gathering information at different levels (e.g., packet inter-arrival time, jitter, packet size, etc.) and analyzing it (e.g., from a statistical point of view), often without inspecting protocol headers and application data transported. Therefore behavioral algorithms are not affected by any of the shortcomings of payload based algorithms related to information hiding (e.g., by encryption) or camouflage (e.g., by using ports typically deployed by specific services). Specifically, behavioral algorithms work the same way independently of whether flows use encrypted payloads or not. Unfortunately, behavioral algorithms have some common limitations; first of all, most of them typically require a pre-classified traffic trace in order to train the classifier before it can start working. These pre-classified traces are usually classified using payload-based methods, manual inspections and human experience; although there are few guaranties about the actual precision of these pre-classified traces, all measurements are done starting from an imprecise base. Furthermore, a wide class of behavioral methods needs to be trained in exactly the same conditions of the environment where they are going to be deployed, which often prevents the training sets obtained in one site from being usable as a trainer set in other places. Additional problems are related to the limited temporal validity of the training set due to network reconfiguration and long term variations, and to the fact that these algorithms often need to observe a fairly large number of packets before they can work properly.

Behavioral algorithms can be further organized into three sub-categories. *Machine learning algorithms* [9] [10] [11] [12] [13] deploy advanced analysis techniques, such as clustering algorithms, to divide network flows in different classes based on information devised without inspecting application layer payload. *Statistical algorithms* [14] process statistical properties of network flows through mathematical function, like Bayesian filters, in order to derive a statistical “fingerprint” for each application. Typical data analyzed by these algorithms are round-trip-time, inter-arrival time, inter-arrival jitter, mean packet size. *Heuristic algorithms* evaluate how each host act within the network in order to identify the applications that hosts are running. Some examples of data analyzed by these algorithms are the order of requests/responses produced by a host, number of hosts contacted, number of ports deployed.

Among heuristic algorithms, BLINC proposed by Faloutsos et al. [8] introduces the idea of looking at the “social” behavior of each host. In fact, the type of traffic a host is producing is devised from the observation of network and transport layer behavior such as how many hosts it contacts or

[1] While the *session table* is usually associated to payload-based techniques, in fact it has a broader usage. Particularly, all methods that rely on session identification (no matter how this identification is done) need to maintain this information in memory.

it is contacted by, how many ports it uses for accepting and producing traffic. The behavioral analysis is carried at *social*, *functional* and *application* level, and contributes to create the so called graphlets graphically describing the behavior of a host. Particularly, graphlets show the number of distinct source and destination ports used, and the number of hosts contacted; the basic assumption is that each application is characterized by a certain type of graphlets. All information needed to create graphlets can be devised from the transport and network layers, thus avoiding application layer payload inspection and its shortcomings and limitations. A classifier that applies this algorithm is first trained by analyzing pre-classified traffic traces to devise application-specific graphlets. Then, new flows can be classified by comparing the corresponding graphlets with the ones obtained from the pre-classified traces. Early classification is obviously not possible with this method because a certain number of interactions must be observed in order to understand which graphlet best represents the behavior of a host. Results look promising, but the solution still suffers from some limitations. Graphlet identification is quite difficult and actually done with human support by observing various types of traffic in order to identify common behaviors within the same kind of application. Moreover, it is difficult to add a new graphlet and making sure it does not overlap with others. Some abnormal behaviors could be classified only by interlacing multiple graphlet and this is not supported by the current solution. NATs are a big problem for this method because they are seen as hosts that summarize the behaviors of all the hosts behind them. Also protocol coverage has drawbacks, since often only the protocol family (e.g. peer-to-peer) can be identified precisely, instead of the actual protocol. Finally, the precision of this method is still lower than the precision offered by methods that use payload inspection for classifying traffic. Summarizing, the idea is interesting, but it is complicated by practical issues involved in handling graphlets.

The idea introduced by Faloutsos et al. that an application has a distinguishable behavior is one of the inspiring principles of service-based classification. However, while BLINC uses specific social behaviors characterizing an application in order to classify the packets of a flow as belonging to that application, the solution proposed in this work relies on the fact that most applications display the specific behavior of offering a service at fixed “network coordinates”, i.e., at a specific port on a specific host. Consequently, once the application providing a certain service is identified, the packets of a flow directed to its “network coordinates” can be classified as belonging to that application. Hence, although the two approaches have a common inspiring idea, the resulting solutions are very different.

An idea similar to our proposal of service classification was already used in previous works, namely [16] and [16], but with some noteworthy differences. [16] proposes to subsequently apply several classification techniques with growing computational complexity, until a flow is classified and to keep a history of already classified flows to build a knowledge base for particular host/port combinations that can be used to *validate* future classification results by checking their conformance with roles previously observed for the same host. In essence, the historical data collected by the solution presented in [16] are not used to classify new flows, but *can be used* to validate the classification outcome of the chain of adopted classification techniques. A similar approach can be found in [16], which proposes a statistical method to classify peer-to-peer traffic; among the three techniques jointly deployed, one consists in keeping a table that contains IP addresses of hosts that are at some point identified as nodes of a peer-to-peer overlay, or that are identified as known (traditional) services (e.g., HTTP server). All flows whose packets contain an IP address included in the P2P table are flagged as “possible P2P” and analyzed in more detail. The main idea is that the type of service a host is currently running can be inferred by looking at the host history (i.e. the sessions generated by the host and the number and type of services contacted in the past). Hence, this approach uses the host history to classify new services, while our proposal relies on other classification algorithms for the initial classification, being the service-based classifier used only in the following classification of that service.

In conclusion, while payload-based methods are usually precise enough and offer excellent coverage (in terms of protocols detected), they are expensive from the memory and computational points of view. Other approaches are promising, but usually limited in terms of coverage, and often suffering from many limitations due to their training requirements. Other methods based on host history are in some sense similar to our work, but they failed to fully understand and exploit the power of service-based identification. The service-based approach presented in this paper is a breakthrough technology that includes as many advantages as possible from both categories while reducing disadvantages. Specifically, the objective is to obtain the high classification precision of payload based methods, while avoiding their limitations related to segmented signatures, encrypted payload, and memory requirements.

III. Service-Based Classification

Service-based classification is a surprisingly simple idea that relies on the observation of how hosts usually interact and on the assumption that certain hosts, usually called *servers*, perform similar interactions, usually offering a *service*, with multiple other hosts over a certain time span. This assumption, which provides the foundation of our method, will be verified through experiments on real network data in Section V.B.

According to the classic client-server paradigm, a potentially large number of hosts connect to a single one to obtain a service. In this situation it is easy to identify the server as a main actor with a long lasting role as it usually offers the same service at the same “network coordinates” (IP address and TCP/UDP port) for a long time. The basic assumption in service-based classification is that knowing which service is offered at an IP address/port pair, a classifier can infer that all sessions directed toward that pair will access such service. For example, if the classifier knows that host `www.polito.it` is running a web server on TCP port 80, it can classify all sessions established to this IP address/port pair as HTTP traffic. It is important to notice that such a classifier does not work like a port based classifier. While the latter assumes that a session is transporting HTTP because it is connected to TCP port 80, a service-based classifier *knows* that `www.polito.it` is running a web server on TCP port 80. When the classifier discovers a service, it stores the triple identifying it — i.e., IP address (of the server), TCP/UDP port (at the server), and transport protocol in an appropriate structure in memory called *Service Table*.

The same principle can be applied to hosts running peer-to-peer applications. In this case the application has a client part and a server part running simultaneously: the client part of a peer establishes sessions to the server part of other peers awaiting for connections at a specific port. How this port is assigned and communicated to the other peers depends on the specific application and protocol, but the key point is that the port used to receive connections from other peers usually does not vary very frequently and is reused many times for the same instance of the peer-to-peer application. So when the client part of a peer connects to the server part of another peer to transfer information, the service-based classifier identifies the server part of such session as a service and stores the associated triple in the service table. Also peer-to-peer applications that use the same port for both the server part and the client part, such as Skype for example, are handled properly. After a peer A has received a connection to its server part, a triple containing its IP address and port is created in the service table as a service. When its client part connects to another peer B, the service-based classifier classifies the corresponding packets according to either A's service entry or B's service entry. Although classification based on A's service entry is in principle mistaken as packets are being exchanged as part of a session whose server side is B, the packets are anyway correctly classified as belonging to the peer-to-peer application at hand. When an application shows such behavior (which is not uncommon among P2P software) our approach can be extended by adding also the client-side of a session to the service table, which will become the server part in a later data exchange, for all traffic belonging to that application.

It is important to notice that finding out which service is running at a certain IP address/port pair (i.e., *service identification*) is orthogonal to the service-based approach: in principle, any method can be used to perform service identification (payload-based, heuristic, or even manual inspection, and more). The service-based approach assumes to know precisely the service associated to an IP address-port couple and from that point on it will guarantee a precise identification of that traffic. Obviously, service identification is not straightforward and its effectiveness has an impact on the outcome of service based classification, as discussed later.

Service-based classification features interesting advantages over other classification methods. *Encrypted traffic* at application layer can be properly classified provided that the corresponding service has been previously identified, i.e., it has an entry in the service table. It offers *pattern segmentation transparency*, i.e., a flow can be properly classified even though protocol identifying patterns are split across multiple packets, avoiding the complexity of reassembling application data units. A service-based classifier needs to maintain only information about services (i.e., IP address, port, transport protocol and service offered) independently of the number of traffic flows actually using such services; hence it has *limited memory requirements*. The limited amount of state information kept by a service-based classifier impacts (*i*) *scalability*, performance in terms of (*ii*) *lookup time* and (*iii*) *hardware implementations* that can rely on faster on-chip memory. Classification of a packet belonging to a known service requires a single lookup on three fields (IP address, port and transport protocol) in a relatively small lookup table, therefore with *low computational cost*. Moreover, service identification, which might have higher computational cost, is expected to be performed only on a small fraction of the packets and it can be even performed offline; in any case, service identification is orthogonal to the service-based method. Finally, as we said, service-based classification is among the few methods that guarantees *early classification*, i.e. being able to classify even the first packet (e.g., a TCP SYN) within each session, while other methods need to process at least the first few packets within each session.

Service-based classification also has some potentially critical issues. Its effectiveness, in terms of minimizing both misses and wrong matches, and also its performance heavily depends on identification of network services that must be as accurate as possible. A wrong entry in the service table leads to wrongly classifying a potentially large number of flows, while a missing entry possibly leads to both a failing classification of a large number of flows and deploying significant amount of computational resources in an effort to identify the service being used, e.g., by deeply inspecting the corresponding packets. Consequently, a successful service-based classification is tightly coupled to a robust and effective service identification solution, which, as we said, is orthogonal to service-based classification.

In addition, not keeping information about individual sessions, service-based classification is not suitable for applications that require such granularity level, such as, for example, per-session enforcement of QoS policies. A service-based classifier can be customized for such applications to keep an additional session table for those services requiring so, which is a simple extension that can be added to any implementation.

Other potential issues include *dynamic sessions* and *proxies*. With respect to the first problem, some applications (e.g. FTP, SIP) use a control session to negotiate the port of the data transfer, resulting in a data-transfer session that cannot be associated to a stable service based on its ports. In order to classify these sessions, a deeper inspection of packets belonging to the control session is required since data transfer is usually made of raw data, making problematic the classification through a common traffic pattern. In order to cope with these sessions, the implementation of a service-based classifier must be able to inspect control packets belonging to these well-known applications, as several modern network devices (Firewall, NAT, etc.) do.

The second problem is related to proxies (and SOCKS servers), which handle the access to various types of services (HTTP, FTP, etc.) on behalf of different clients using the same transport-layer port. The service-based classifier has no problem in classifying traffic from these servers to the target service, but it is unable to distinguish the service contacted when analyzing the traffic between clients and the proxy/SOCKS server, which can be done only by application data inspection. Like for services that use dynamic sessions, the service-based method can be integrated with a callback function that performs deep inspection on packets exchanged between clients and their proxy and socks servers in order to identify each flow accessing a specific service.

Finally, the service-based classifier is not effective with traffic encrypted at IP level, e.g. with IPsec.

IV. Implementing a Service-Based Classifier

Although the service-based classifier looks simple and elegant, some issues need to be addressed to make it working properly. This section presents such issues and gives some insight in how they have been addressed in our implementation. Given the generality of the service-based method, other implementation strategies can be adopted.

A. *Service identification*

Given the expertise and previous work of the authors, a payload-based implementation of a service identification module has been an obvious choice. In particular, an existing packet processing engine based on the Network Packet Description Language (*NetPDL*) [1] [5] has been reused in the implementation of the service identification module. NetPDL is an application-independent packet format description language that enables the creation of a generic protocol description database: the NetPDL database, in fact. Although it includes only packet header formats and does not support the description of protocol temporal behavior (e.g., a protocol state machine), it has proved being extremely effective and robust with respect to traffic classification [4], thanks to an extension that enables management of lookup tables, originally used to maintain transport-level sessions [5]. The high flexibility of NetPDL makes the engine suitable for the implementation of the service-based classifier as well, in addition to the payload-based service identification module.

The main modification made to the NetPDL engine is the addition of some new tables, such as the service table that contains information about services. The process starts with an empty service table, while traffic is processed by extracting IP addresses and ports from each arriving packet. Since the server side of the communication cannot be inferred on a packet-basis, the service table is looked up twice: once with the source identification (source IP/port) and once with the destination identification. If one of these lookups is successful, the packet is classified through the service-based method. Otherwise, as depicted in Figure 1, the service identification module performs a payload-based classification to possibly introduce a new entry in the service table containing the IP address and the transport layer port used by the server side of the session and the application protocol associated. Any new packet toward this "known service" can subsequently be classified directly through the information kept in the service table as described above without any further processing (e.g., payload inspection). Please note that the identification of the server side of the connection is not straightforward and will be discussed in Section IV.B. As time passes, more and more traffic will be classified by the service-based method since the service table will include an increasing number, possibly most, of the services active in the network.

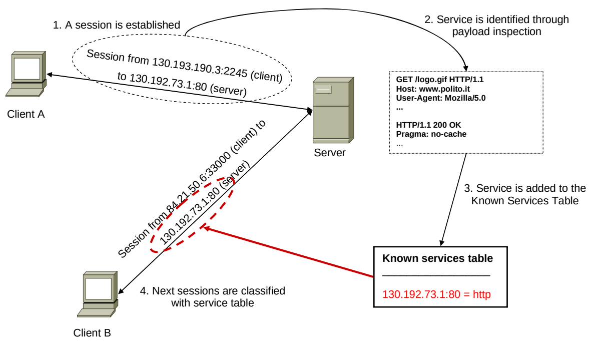
1. A session is established
Session from 130.193.190.3:2245 (client) to 130.192.73.1:80 (server)
Client A
Server
2. Service is identified through payload inspection
GET /logo.gif HTTP/1.1
Host: www.polito.it
User-Agent: Mozilla/5.0
...
HTTP/1.1 200 OK
Pragma: no-cache
...
3. Service is added to the Known Services Table
Session from 84.21.50.6:33000 (client) to 130.192.73.1:80 (server)
Known services table
130.192.73.1:80 = http
4. Next sessions are classified with service table
Client B

Figure 1. Service identification.

B. *Distinguishing clients and servers*

The server side of a TCP session can be easily identified by observing the SYN and ACK flags during in the three-way handshake of the TCP protocol. In our implementation we use an additional lookup table, called *Candidate Service Table*, in which a new entry is added with the IP address and port of a host that accepted an unclassified TCP session by generating a TCP packet with both the SYN and ACK flag enabled. The Candidate Service Table is required to keep track of the server side of a session because the service is possibly identified, e.g., through payload inspection, once the session has been opened, i.e., when the SYN/ACK flags, used only during the initial handshake phase, are not available to enable the identification of the server side. When the service is finally identified, the server information is moved from the candidate service table the service table.

Entries of the Candidate Service Table are subject to a very fast ageing (about ten seconds [19]) in order to avoid their number to explode over time due to sessions opened by unidentified services, unsuccessful handshakes, or unused opened sessions, as in cases of malicious activity such as SYN flooding and port scanning.

With UDP services identifying the server is different since explicit information like the SYN flag in the TCP case is not available. Although, especially with the growing adoption of broadband multimedia applications, UDP is expected to significantly increase its traffic share, possibly becoming predominant, this paper focuses on TCP traffic, which as of today accounts for the vast majority of data. UDP traffic classification, that requires a non-straightforward extension of what is proposed in this work, is left to a companion future paper.

C. *Managing the service table*

Besides properly populating the service table, an important issue is the prompt elimination of service entries once the corresponding service is no longer provided. This is important in order to avoid the explosion of the number of service entries and that a service offered only temporarily leads to classification errors. One possible approach is to purge an entry that does not make a hit for a certain amount of time, hereafter referred to as *service inactivity timeout*. As a further refinement, the service inactivity timeout can be differentiated for different service classes. For example, some services are offered over a long time period, possibly permanently, even with a low connection rate, and their entries are given a long service inactivity timeout. A typical example of this service class is an SMTP server contacted only few times in a day, but providing its service over a very long time period. Vice versa, other services have a naturally short life and the inactivity timeout associated to their entry may be shorter. Typical examples are peer-to-peer applications.

Assigning distinct service inactivity timeouts to different classes of services, although not strictly necessary, is useful in avoiding multiple re-identification of long-term services, e.g., through costly deep packet inspection. On the other hand, assigning an entry to the long-term service category is critical because if the service is not actually long-term or it has been wrongly identified, the entry can lead to persisting classification errors. Consequently, there should be a certain level of certainty about service before categorizing it as a long-term one. One possible policy is to set any newly identified service "under observation": its entry is categorized as short-term and some additional checks are performed on packets classified according to the entry. For example, payload inspection can be executed on randomly chosen new sessions. After a certain period of observation confirming the initial identification, hence the long-term nature of the service, the corresponding entry can be categorized as long-term. Another policy can be to categorize services as long-term only through explicit (e.g., manual) configuration.

With respect to the scalability of service based classification, it is worth noticing that the management of the service table is independent of the classification process and can be implemented as a distinct process running separately from the core classification process.

V. Experimental Evaluation

This Section provides an experimental evaluation of service-based classification, including some problems that arise in its implementation. The next section first devises the benefits expected by the deployment of service-based classification from an analysis of network traffic itself — i.e., not based on the results of particular classification experiments — which provides a more general assessment of the potential of service based classification. Then, the results of specific classification experiments are reported to substantiate such general assessment.

A. *General Assessment*

Before implementing our service-based classifier we collected a set of session-related statistics on the link that connects our University to the Internet to assess the potential benefits of service-based classification in terms of memory occupancy, i.e., if the number of services was really smaller than the number of sessions. These measurements, done using *Tstat* [15] and lasting several days, wanted to determine the maximum number of service entries required to classify all the traffic with a service-based approach, compared to the number of session entries required by a classifier based on session identification. The obtained results must be intended as a lower bound of the session/service table size since they account for the session/services present and actually active at any given time. A TCP session is considered closed when a FIN or RST packet is observed; in case of abnormal termination, a 10-minutes session inactivity timeout is used to declare a session terminated, as suggested in [22] and 0. Analogously, services are considered closed if no traffic is observed in an idle period of the same duration.

Figure 2 shows, for each minute, the number of active traffic sessions and the corresponding number of services on the uplink (100 Mbps) of our university network (about 6,000 hosts) over a 7-day period. The average number of active traffic sessions is 80,000 with peaks of 180,000, while the total number of services never exceeds 10,000. Figure 3 shows the same figures for a traffic trace[2] from the MAWI wide traffic archive [21]. The average number of active session is 120,000 with a peak of 380,000, while the total number of services never exceeds 10,000. The average on the whole observation period of the *session to service* ratio is about 20 for both traces, which means that a service table requires roughly 20 times fewer entries than a session table. Furthermore, a service entry is smaller than a session entry, thanks to the smaller number of information that has to be stored. This is beneficial in terms of memory requirements as well as both processing requirements and performance for session/service information look-up.

Although these numbers show clearly the advantage of the service-based classification (at least in the tested

[2] This trace was captured on March 19th, 2008 on the sample-point F, a 150Mbps trans-pacific backbone link.

environments[3]), they are derived under the assumption that services are stable over time. This assumption will be empirically demonstrated in Section B.

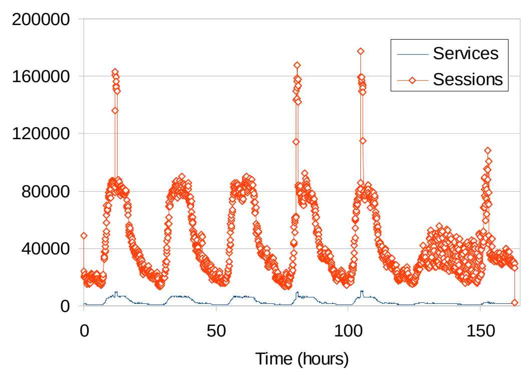

Figure 2. Services vs. sessions on the University Torino network.

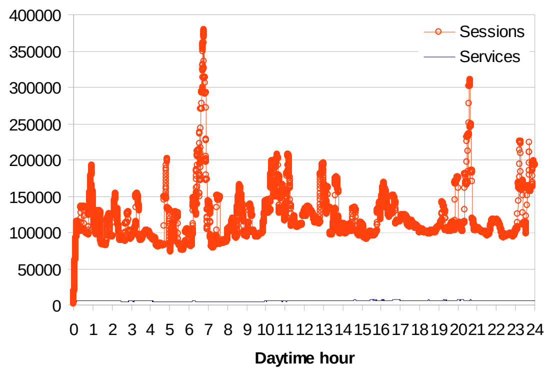

Figure 3. Services vs. sessions on a trans-pacific link.

a) Experimental setup

The service-based classifier implementation described in Section IV has been tested in different experiments; the results obtained with two of them (more details on these experiments are shown in Table 1) are presented in this paper. The first experiment contains three days of traffic related to a single subnet of our University during a week-end (hereafter called *Weekend* trace). The second one is a twelve hours analysis of the traffic on the whole campus network during a working day, hereafter referred to as *WorkingDay* trace. The *Weekend* trace is particularly challenging from the classification point of view because it contains mainly eDonkey peer-to-peer traffic that is known to be difficult to classify; *WorkingDay*, instead, includes mainly HTTP traffic. Although our traces contain all kinds of traffic, as previously discussed, this work focused on TCP sessions only. Traffic was processed on an Intel Dual Xeon (2,4GHz, 512KB cache, 1GB RAM, 4 disks for an aggregated space of 900GB) equipped by an Endace DAG card for packet capture, ensuring that no packets were dropped during the capture process. Traces were analyzed with both a payload-based classifier based on the NetPDL described in Section IV.A and a service-based classifier that uses the same payload-based classifier for service identification as described in Section IV. The comparison of the results obtained with the two classifiers shows benefits in terms of both memory usage and classification precision when service-based classification is combined with an existing classifier. Given that our service-based classifier implementation is not optimized for performance, it cannot be meaningfully deployed for an assessment of classification rate, which in any case strongly depends also on the underlying hardware. Such important assessment is left for future work.

[3] The gain guaranteed by the service-based method may be more limited in case of different network conditions. We may speculate that the *session vs services* ratio may be smaller in case of a network with large percentage of P2P traffic (e.g. an ISP POP). However the authors are not aware of any publicly available (and recent) trace to verify this speculation.

TABLE 1. SUMMARY OF THE TRACES USED IN OUR ANALYSIS

Trace	Description
Weekend	65 hours trace (from 11/05/2007 – 2.00 pm to 14/05/2007 – 7.00am), 89 hosts in the internal network, 66M packets (69% TCP), 35 GB traffic (86% TCP)
WorkingDay	12 hours trace (from 11.00am to 11.000pm on 20/12/2007), 5649 hosts in the internal network, 583M packets (88% TCP), 465 GB traffic (95% TCP)

B. *Service stability*

As mentioned in Section III, the fundamental assumption of the service-based classifier is the stability of services. In fact, a service-based classifier can misclassify packets when either (*i*) a service has not been properly identified or (*ii*) a new service is offered at the same "network coordinates" where another service was previously offered. The last phenomenon is related to service stability. Given that (i) is a shortcoming of the classifier used for service identification and is orthogonal to the service-based classifier itself, service stability is the key factor impacting the accuracy of service-based classification.

In order to prove this assumption, we used a tool jointly developed at University of Brescia and Politecnico di Torino, which installs a probe in each host and logs on a centralized server the name of the application that created each network socket on the host on which it is running. By deploying it on all the hosts of a network used for the evaluation of a classifier, it is possible to precisely know which application generated each session. The tool has been installed on 11 hosts (with Linux, Windows and MacOS-X operating systems, running several applications; among the other Skype, Emule, Joost, uTorrent), the traffic produced has been captured for 4 days and the traffic traces have been analyzed by a payload-based classifier.

TABLE 2. APPLICATION MONITORING VS. SERVICE-BASED CLASSIFICATION

Observed sessions	40503
Observed services	21675
Observed applications	81
Services in which sessions are classified univocally as belonging to the same	21042

application	
Services in which sessions are classified univocally as belonging to the same application or as "unknown" traffic	633
Services in which sessions are classified as belonging to different applications by the payload-based classifier (or by manual inspection)	0

Table 2 shows that the payload-based classifier identified the vast majority of the sessions associated to a given service as belonging to the same protocol. In a small minority of cases, some sessions were unclassified, mostly due to signature-related problems (e.g., a signature split across two packets), while in a very limited amount of cases we found different classification results associated to the same services. In the latter case we further analyzed these sessions with a manual inspection and we verified that these mismatches were due to an error of the payload-based classifier. These results verified that classification results produced by the payload-based classifier are always coherent with the application that created. The overall result was that we found no cases in which sessions belonging to different protocols were referring to the same service, which provides a strong (albeit experimental) foundation to our method. In other words, since all the services analyzed do not change during their service time, it is possible to state that the classification error introduced by a service change during its lifetime is, to say the least, really rare.

C. *Service activity and service inactivity timeout*

Service lifespan must be taken into consideration in the management of the service table, i.e., in the choice of the service inactivity timeout. On the one hand, the larger the number of services listed in the table (as it can be obtained with a long service inactivity timeout), the larger the amount of traffic possibly classified by the service-based classifier. On the other hand, a long service inactivity timeout may lead to a dramatic increase of the size of the service table, reducing the scalability of service-based classification. Moreover, keeping entries in the service table for a long time amplifies the impact of service identification errors as each misclassification (e.g., a pattern mismatch occurs when identifying the protocol deployed by the service) potentially impacts the classification of all packets sent and received at the corresponding network coordinates (i.e., IP address/port pair). It is worth mentioning that identification error probability can be reduced by improving the deployed service identification method, which is independent of service-based classification. Also, multiple classifiers with different properties (i.e., strengths and shortcomings) could be used in parallel to provide service identification. These aspects, although significant and relevant to the overall performance of the classification process, are outside the scope of the current work. Instead this Section aims at studying the activity level of services to gain a better understanding on how to set the service inactivity timeout.

Based on the authors' experience in analyzing network traces, service can be categorized in three classes with respect to their activity level and *lifespan*, i.e., the overall time frame during which they show some activity. *One-shot services* are active for a very short time (a second or less), and are never re-contacted. Other services, named *intermittent services*, are active repeatedly very shortly each time; these services are usually contacted by few hosts rarely and for short time periods and their sessions are somewhat distributed along the entire trace, i.e., they have a rather long lifespan. *Continuous services* are characterized by rather long activity periods, are usually contacted frequently, and are expected to generate most of the traffic. Figure 4 provides a graphical representation of the activity time of some representative services; each horizontal line represents the time period when the service is active. Classes with ID from 1 to 6 are continuous services, with a long period of activity without breaks. Services with ID from 7 to 12 are one-shot services since their activity time is very short and they do not reappear. The last group of services, with ID from 13 to 18, are intermittent services.

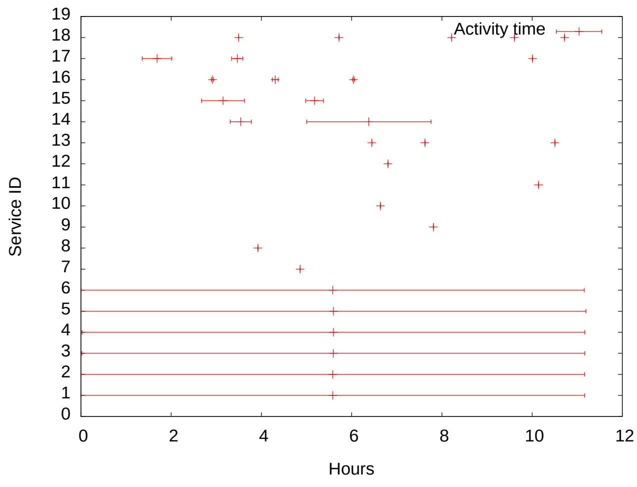

Figure 4. Graphical representation of per-class service activity for trace WorkingDay.

Intermittent services are the most critical with respect to setting the service inactivity timeout. If their inactivity periods are longer than the service inactivity timeout, they have to be re-identified each time they become active again, thus introducing overhead and reducing performance.

In order to characterize service duration and have precise information about the lifespan of each service, we modified the service based classifier in order to log the insertion and deletion of services in the service table, which will provide precise information about the lifespan of each service. Figure 5 shows a histogram of the lifespan of services in trace *WorkingDay* highlighting the percentage of services that have been identified when running a service-based classifier with two different values of service inactivity timeout: 10 minutes and 60 minutes[4]. Some bars are made up of a continuous portion and a dashed portion: the dashed portion takes into account of the services that have already been identified in the past and that appeared again after the service inactivity timeout. As shown in Figure 5, the vast majority of the services belonging to the *(30-120] seconds* category (with inactivity timeout of 10 minutes) lasted for some time, then disappeared from the trace for more than 10 minutes, then appeared again in the trace, although their total lifespan (including their inactivity time) falls within this bin. In this example, intermitted services belonging to the (30-120] bin accounted for about 6% of the total amount of observed services and are by far the vast majority of services belonging to that bin. Our analysis confirms that overall 41% of services have to be re-identified with a 10 minute service inactivity timeout versus 31% with a 60 minute service inactivity timeout. In summary, the 60 minute service inactivity timeout provides little improvement as far as service re-identification — mostly concentrated among services with lifespan between 30 and 120 minutes, as shown in Figure 5 — and a significant percentage of services with a large lifespan (greater than 120 minutes) still requires re-identification. On the other hand, the 60 minute service inactivity timeout has a serious impact on memory usage (as demonstrated in Section V.D), since many services that will never have traffic again are retained in the service table for a long time.

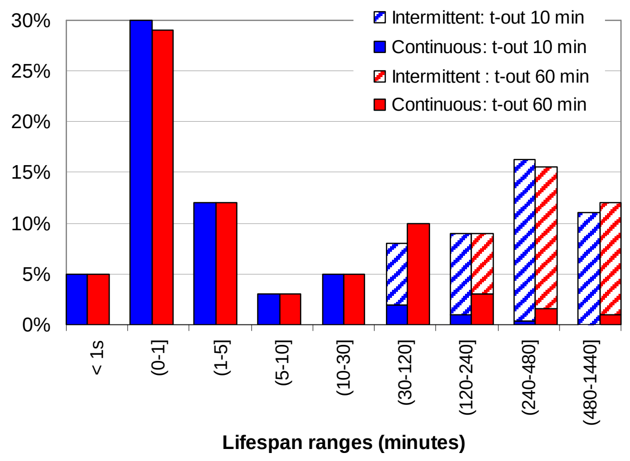

Figure 5. Service lifespan distribution for trace WorkingDay

In order to limit the performance impairments caused by intermittent services with long lifespan and rather short activity periods, service table entries could be assigned to different classes, each characterized by a specific service inactivity timeout, as discussed in Section IV.C. Analysis over longer periods of time and a wider range of network scenarios would be required to further substantiate what has been presented here and provide an insight on long term service change dynamics. However, the experiments reported in this section demonstrate the importance of the service inactivity timeout and their outcome can be used as a general guideline for an algorithm to dynamically adjust the inactivity timeout over time depending on the service type.

D. Ageing of service table entries

As shown in the previous section, the service inactivity timeout associated to service table entries can impact the performance of the classifier. We analyzed several traces with a payload-based classifier and with a service-based one configured with a service inactivity timeout of either 10 minutes or 60 minutes; Figure 6 shows the classification results obtained on trace *WorkingDay*. No significant differences can be observed in terms of classification results with the two service inactivity timeout configurations: the overall number of packets classified based on a service table varies from 81% to 85%, which is a reasonably small improvement. Furthermore, the variation in terms of unclassified traffic is negligible, varying from 4.76% to 4.45% (in terms of number of packets) when changing the value of the service inactivity timeout[5]. This means that a 10 minute aging time for service table entries is a good trade-off because it to assures high performance and low memory requirements. Moreover, it means that the payload-based classifier does its job nicely and it is able to re-classify new sessions that do not have a corresponding entry in the service table.

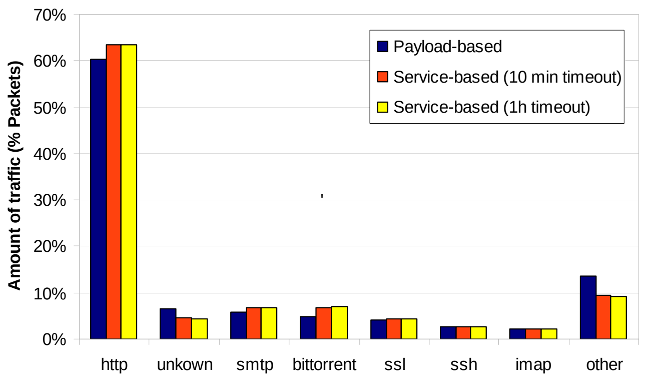

Figure 6. Classification results on trace WorkingDay.

An important observation is that simply increasing the service inactivity timeout may not be a good idea, since we may end up filling the service table with entries related to one-shot services or services that are anyway not any longer active, which will never appear again in the future. This is evident in Figure 7 that shows an almost four-fold increase of the service table size when changing the service inactivity timeout from 10 to 60 minutes— without any appreciable advantage in terms of classified traffic, as shown by Figure 6. Therefore, a 10 minute service inactivity timeout has been used in the experiments producing all the results presented in

[4] The sum of intermittent and stable services of each bin should not differ irrespective of the inactivity timeout used. This would be true only if the payload based classifier used to identify services is able to classify services analyzing any session produced by them. Unfortunately this is not true, especially for services that produce encrypted sessions. Thus some services are missing in some bins because the classifier was not able to classify them when they reappeared after an idle period longer than the inactivity timeout.

[5] The increase in the amount of classified traffic when using larger timeout will be explained in Section V.E.

the rest of the paper.

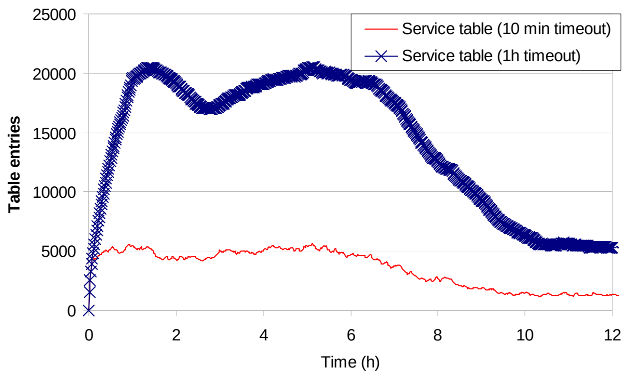

Figure 7. Service table size with different service inactivity timeouts on trace WorkingDay.

It is worth noticing that the vast majority of the traffic is generated by a fraction of all services. It may be an interesting idea to keep only such services in the service table, which enables to classify most of the traffic. Figure 8 shows the cumulative quantity of traffic generated by each service in both the *WorkingDay* and *Weekend* trace, where services have been sorted from in order of decreasing amount of generated traffic. We can see that, assumed equal to 1 the amount of traffic that we are able to classify with the service-based method (using a 10-minutes timeout and a payload-based classifier for detecting the service once it appears), we are able to classify 90% of that traffic with only 3.8% of the number of services present in the entire *WorkingDay* trace (3344 out of 87815), and 2.7% in case of the *Weekend* trace (338 out of 12714). These numbers can also bring to the conclusion that, once identified the most important service contributors within the network, these services alone enable the classification of the vast majority of the traffic. Obviously, this has a major impact also on scalability, which is greatly improved by limiting the service table to only the most verbose services. While these numbers are somewhat expected (for instance, the splitting of traffic sessions among mice and elephants is well known in the literature [20]), in our case this represents an even more important gain. For instance, sessions terminate after a relatively limited time, while services may stay stable for month or years (e.g., the Google website), hence the service table can even contain entries that must be verified only occasionally.

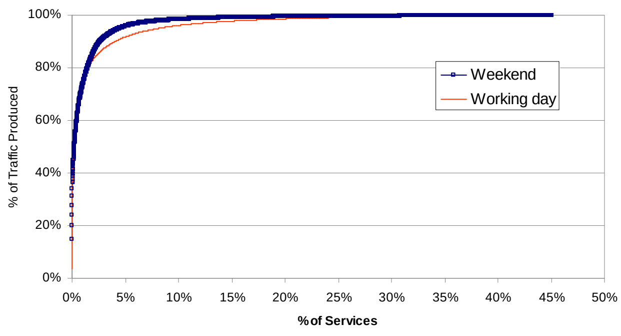

Figure 8. Cumulative classified traffic; services are ordered starting from the one that generates the most part of the traffic.

E. Accuracy

Our tests show that service-based classification offers an improvement in classification accuracy over results obtained with the original payload-based classifier. For example, trace *Weekend* contains a significant amount of traffic generated by eDonkey that hinders payload-based classification when application-layer data is encrypted. The payload-based classifier recognized only a small percentage of the flows generated by these applications, e.g., some sessions that are occasionally sent in clear and that represent special cases. For example, Skype sometimes produces packets that are only partially encrypted and consequently can be properly inspected and classified; similarly, not all eDonkey messages are encrypted. In all the other cases, the payload-based classifier is unable to identify the protocol transported and it marks flows as unknown, as it is shown by the high percentage of unknown traffic in Figure 9. Experimental evaluation also showed another problem related to the completeness of the pattern database used by the payload-based method. In fact, some unknown traffic is related to flows that use particularly rare or undocumented application level messages that are not part of the pattern database of the payload-based classifier. Service-based classification does not have this problem, because once a service has been identified thanks to the presence of some known signatures in application-level messages, following sessions are classified based on the network coordinates they are related to. This is confirmed by Figure 9 where the service-based classification leaves a much smaller amount of traffic as unknown, while classifies as eDonkey a much larger percentage of traffic than payload-based classification. Results reported in Figure 9 are referred to the percentage of packet classified; results are slightly worse in terms of bytes, in which the percentage of the unknown traffic is 11%.

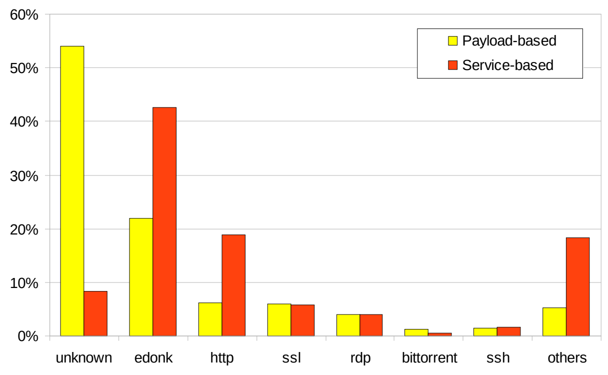

Figure 9. Classification results of the Weekend trace with payload-based and Service-based method.

Payload based classification on trace *WorkingDay* results in a low percentage of unknown traffic because the trace includes mainly HTTP traffic. However, service-based classification results in improved accuracy also on the *WorkingDay* trace. Figure 10 focuses only on the unclassified traffic of Figure 9 and shows how this traffic has been classified by the service-based classifier. For instance, among the 54% of unclassified traffic of the *Weekend* trace, about 18% was eDonkey, 14% RPC (which is included in the "others" bin in Figure 9), and more. Manual investigations on a randomly chosen subset of classified flows confirm that the outcome of the service-based classifier is correct.

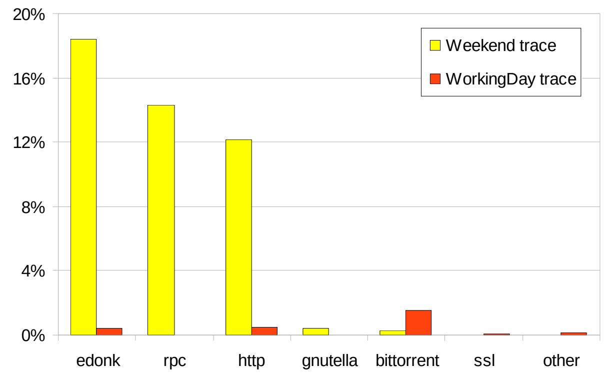

Figure 10. Improvement in classification accuracy; packets unclassifed by payload-based classifier and succesfully classified by the service-based classifier.

While the improvements in terms of unclassified traffic are evident from previous graphs, a deeper analysis shows also that a non-negligible portion of the traffic classified by the payload-based classifier resulted wrong when re-classified with the service-based method, mainly due to applications that exploit tunneling techniques (e.g., Edonkey traffic encapsulated in HTTP). Figure 11 shows the percentage of packets that are classified differently by the payload-based classifier and the service-based one. More specifically, for each protocol X, positive values represent the percentage of packets that the service-based classifier assigns to X given that they were classified differently with the payload-based method ($\Delta_{SBC}((C_{SBC} == X) \mid (C_{PBC} != X))$). Negative values represent the amount of packets that were classified as belonging to X by the payload-based classifier, and that resulted no longer belonging to that protocol with the service-based method ($\Delta_{SBC}((C_{SBC} != X) \mid (C_{PBC} == X))$). In other words, being $T_{SBC}(X)$ the amount of traffic belonging to protocol X as classified by the service-based classifier, and $T_{PBC}(X)$ the amount of traffic classified by the payload-based as belonging to protocol X, we have:

$$T_{SBC}(X) = T_{PBC}(X) + \Delta_{SBC}((C_{SBC} == X) \mid (C_{PBC} != X)) - \Delta_{SBC}((C_{SBC} != X) \mid (C_{PBC} == X)) \qquad (1)$$

A precise analysis of the results shown in Figure 11 is problematic because, for sessions classified differently, we do not know which method returns the correct result. A manual inspection of some of these sessions (randomly chosen) showed that the vast majority of them were correctly classified by the service-based method; therefore we assume that the results of the service-based classifier are correct in the following discussion[6]. For trace *Weekend*, eDonkey sticks out as the most problematic protocol as more than 2% of the total amount of packets is not correctly classified by the original classifier, while for trace *WorkingDay*, HTTP and Bittorrent have the most significant values. Commenting these results is rather difficult; we can take the HTTP traffic present in the *WorkingDay* trace as an example. In this case, a deeper inspection shows that most of the false negatives of the payload-based classifier are due to the HTTP header split across several packets (e.g., related to new interactive services characterized by long URLs). In the meanwhile, false positives are mainly due to Bittorrent that was misclassified as HTTP due to its similarity in the application-layer header. The fact that values in Figure 11 are relatively small (e.g., compared to the unclassified traffic analyzed in Figure 10) shows that there are few disagreements between the two classifiers, i.e., both classifiers make few classification errors.

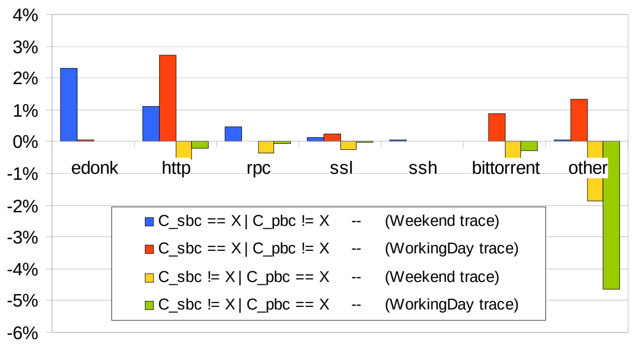

Figure 11. Migration of classification result.

F. *Scalability*

Scalability must be assessed in terms of memory and processing requirements.

[6] If this assumption holds, the second term of Equation (1) represents the false negatives of the payload-based classifier, while the third term represents the false positives.

From the memory standpoint, the number of entries required by the service table is more than one order of magnitude smaller than the number of entries required by the session table, as shown in Figure 2 and Figure 3. Moreover, each service entry is approximately half the size of each session entry. This large improvement is not mitigated by the necessity of some additional information such as the candidate service table, whose size is negligible (Figure 12). In fact, Figure 12 shows that the additional table required to keep track of the potential services (as presented in Section IV.B) is reasonably small (in average, one tenth of the service table), hence has a negligible impact on the memory requirement of the service-based classifier. Figure 12 shows also the number of entries in the service table when processing the *WorkingDay* trace. The average number of 5000 entries during daytime hours fits well with theoretical lower bound derived from Tstat analysis and shown in Figure 2 and Figure 3. This confirms the goodness of our implementation of the service-based classifier.

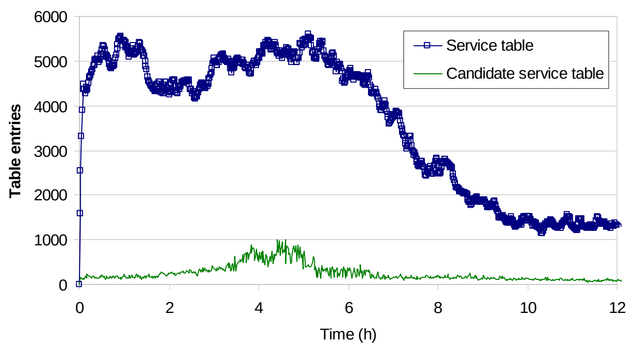

Figure 12. Entries in the service and candidate service tables.

From the processing side, the computational complexity of a classification solution is an important index of its scalability. Profiling done on our classification code (written in C/C++) confirmed that the cost for a lookup in the service table (i.e. the main cost associated to each packet by the service-based method) is 37 times lower than the cost for a pattern matching on the payload (9700 clock ticks against 260[7]). Although the asymptotic processing cost remains the same in both service-based and payload-based classifier (in the unfortunate case in which each service is associated to a single session), in practical terms our method guarantees a speed-up of more than an order of magnitude at best.

In summary, the performance and scalability improvements of service-based classification over payload-based classification is directly proportional to the percentage of traffic classified by the service table, i.e., without performing payload inspection. Figure 13 shows that the average of such percentage is between 85% and 90% of the total number of packets for the *Weekend* trace, which is coherent with the results shown in Figure 9[8]. This number becomes closer to 100% (usually between 95% and 99%) when considering only TCP packets with payload, which means that the vast majority of traffic unclassified by the service-based method is related to very short sessions directed to previously unknown services (i.e., a typical pattern for peer-to-peer applications), in which the 3-way TCP handshake accounts for the most part of the traffic. Similar results were derived from the *WorkingDay* trace, even if they are really different in term of time period observed and number of hosts considered.

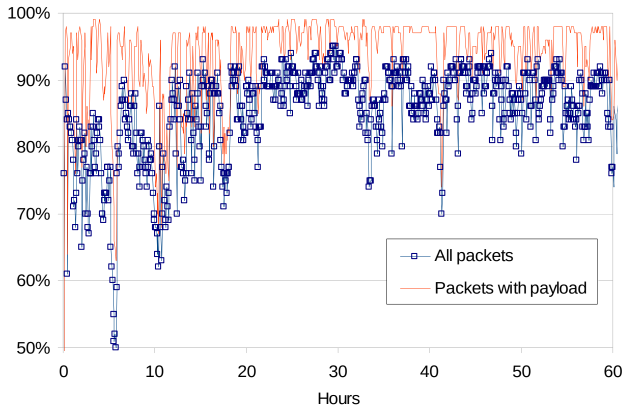

Figure 13. Percentage of packets classified with the service-based method (Weekend trace).

VI. Conclusions

This paper presents a new idea for traffic classification, named *service-based classification*, that is, in some respect, orthogonal to the other classification techniques. This method introduces also in the traffic classification arena the concept of *fast path*, through which the vast majority of the traffic is processed with a limited use of processing and memory resources —ultimately in a short time— and a *slow path* that is invoked in a limited number of cases. In this respect, service-based classification aims at providing a solution to the fast-path processing by proposing that traffic be classified according to the *service* it belongs to. A service is identified by a *serviceID*, which is the tuple {*server IP address, transport-level port, protocol*}. Experimental data confirm that services are very stable even over long periods, making this method extremely simple, efficient and robust. Particularly, robustness is achieved because this method does not require the analysis of all sessions: provided that a service has been previously recognized, sessions accessing it can be classified even if encrypted at application-layer or data flow is

[7] The pattern matching is calculated as the average cost for applying a regular expression on a packet that matches the input string itself, using a modified Boyer-Moore algorithm. Costs are significant higher (some order of magnitude) for non-matching packets and using the DFA algorithm. Measurements have been done using the `pcre` library. Lookup costs have been derived by creating a lookup table through the `stl::map` C++ STL template, which is implemented as a binary tree, and creating a table with 50K entries.

[8] The total number of packets is the sum of packet classified by the service table (about 85%), the unknown traffic (about 9%), and the ones that are related to the TCP initial handshake plus the first packet(s) of the sessions that are classified according to the payload-based method. Once the session is classified by the payload-based module, following packets are included in the amount of traffic classified by the service table.

observed only in one direction. Results in terms of efficiency are impressive, leading to a 37x reduction in processing cost, and a 20x reduction in the number of entries in data structures compared to session based classifiers at least in the traffic trace examined; furthermore each entry being half the size. Real-time measurements on the actual traffic transmitted on the upstream link of our University show that roughly 81% of the packets and 93% of the traffic (in terms of bytes) is successfully classified with the proposed method. Furthermore, service based classification is among the few methods that guarantee early classification, including the initial TCP handshake of a session. Among the few drawbacks of this method is the impossibility to classify IPsec traffic. It is worthy noticing that the precision of the service identification process is crucial for obtaining high-quality results, since a mismatch in service identification will impair the classification of all the sessions related to that service.

Thanks to its fast-path approach, service-based classification provides a way of making the deployment of sophisticated methods for service identification, such as statistical or behavioral algorithms, or even a combination of them with payload-based classification, practical notwithstanding their high complexity and processing requirements. In fact, by using such solutions to discover network services and hence populate the service table, i.e., only on the slow path, their complex algorithms are executed only on a limited number of packets. From a certain point of view, the efficiency of the service-based approach frees computation and memory resources that can be used to identify more precisely the services. Finally, it is also possible to adopt a service database built offline, possibly provided by a third party and modeled after the signature database of antivirus programs.

REFERENCES

[1] Computer Networks Group (NetGroup) at Politecnico di Torino. *The NetBee Library*. August 2004. [online] Available at http://www.nbee.org/.

[2] S. Sen, O. Spatscheck, D. Wang. Accurate, scalable in-network identification of p2p traffic using application signatures. *Proceedings of World Wide Web Conference*, pp. 512-521 NY, USA, May 2004.

[3] P. Haffner, S. Sen, O. Spatscheck, D. Wang, D. 2005. ACAS: automated construction of application signatures. In *Proceedings of the 2005 ACM SIGCOMM Workshop on Mining Network Data*, pp. 197-202, Philadelphia, USA, August 2005.

[4] F. Risso, A. Baldini, M. Baldi, P. Monclus, O. Morandi. Lightweight, Session-Based Traffic Classification. *Proceedings of the IEEE International Conference on Communications* (ICC 2008) - Advances in Networks & Internet Symposium, Beijing, China, May 2008.

[5] F. Risso, A. Baldini, F. Bonomi. Extending the NetPDL Language to Support Traffic Classification. In *Proceedings of IEEE Globecom 2007*, Washington, D.C, USA, November 2007.

[6] R. Pang, V. Paxson, R. Sommer, L. Peterson. Binpac: a yacc for writing application protocol parsers. In *Proceedings of the 6th ACM SIGCOMM on Internet Measurement*, pages 289-300, Rio de Janeiro, Brazil, October 2006.

[7] O. Reviv. Inside network programming with SML. *EE Times*, August 2003. Available at http://www.eetimes.com/story/OEG20030818S0077

[8] T. Karagiannis, K. Papagiannaki, and M. Faloutsos. BLINC: Multilevel Traffic Classification in the Dark. In *Proceedings of ACM SIGCOMM*, pages 229–240, Philadelphia, PA, August, 2005.

[9] A. Este, F. Gringoli, L. Salgarelli, Machine Learning techniques for traffic classification: an approach based on Support Vector Machines. *Technical Report*, November 2007.

[10] J. Erman, A. Mahanti, M. Arlitt. Traffic Classification using Clustering Algoritms. *Proceedings ACM SIGCOMM Workshop on Mining Network Data (MineNet 06)*, Pisa, Italy, September 2006.

[11] J. Erman, A. Mahanti, M. Arlitt, C. Williamson. Identifying and Discriminating Between Web and Peer-to-Peer traffic in the Network Core. *Proceedings of the 16th International World Wide Web Conference (WWW)*, pp. 883-892, Banff, Canada, May 2007.

[12] N. Williams, S. Zander, G. Armitage, Evaluating Machine Learning Algorithms for Automated Network Application Identification. *CAIDA Technical Report* 060410B, April 2006.

[13] T.T.T. Nguyen, G. Armitage. A Survey of Techniques for Internet Traffic Classification using Machine Learning. To appear in *IEEE Communications Surveys & Tutorials*, (4th edition 2008).

[14] M. Crotti, M. Dusi, F. Gringoli, L. Salgarelli. Traffic Classification through Simple Statistical Fingerprinting. *ACM SIGCOMM Computer Communication Review*, Vol. 37, No. 1, pp. 5-16, Jan. 2007.

[15] M. Mellia, A. Carpani, R. Lo Cigno. TStat: TCP STatistic and Analisys Tool. *Proceedings of the 2nd International Workshop on Quality of Service in Multiservice IP Networks* (QoSIP2003) - LNCS2601, Milano, Italy, February 2003.

[16] A. W. Moore, K. Papagiannaki. Toward the Accurate Identification of Network Applications. *International Workshop on Passive and Active Network Measurement* (PAM 2005), Boston MA , USA, vol. 3431, pp. 41-54, March 2005

[17] T. Karagiannis, A. Broido, M. Faloutsos, Kc claffy.Transport layer identification of P2P traffic. *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement table of contents,* pp. 121 - 134, Taormina, Italy, Oct. 2004.

[18] G. Varghese, J.A. Fingerhut, F. Bonomi. Detecting Evasion Attacks at High Speeds without Reassembly. *Proceedings of ACM SIGCOMM 2006*, Pisa, Italy, September 2006.

[19] H. Kim, J.-H. Kim, I. Kang, S. Bahk. Preventing Session Table Explosion in Packet Inspection Computers. *IEEE Transactions on Computers*, vol. 54, no. 2, pp. 238-240, February 2005.

[20] C. Estan, G. Varghese. New Directions in Traffic Measurement and Accounting: Focusing on the Elephants, Ignoring the Mice. *ACM Transactions on Computer Systems*, vol. 12, issue 3, pp. 270-313, Aug. 2003.

[21] Measurement and Analysis on the WIDE Internet Working group traffic archive, *http://tracer.csl.sony.co.jp/mawi/*

[22] N. Brownlee. Traffic flow measurement: Meter MIB. Request for Comments RFC 2064, Internet Engineering Task Force, January 1997. Cooperative Association for Internet Data Analysis, *Network Traffic Measurament Tool* http://www.caida.org/tools/measurement/netramet/

First A. Author (M'76–SM'81–F'87) and the other authors may include biographies at the end of regular papers. Biographies are often not included in conference-related papers. This author became a Member (M) of IEEE in 1976, a Senior Member (SM) in 1981, and a Fellow (F) in 1987. The first paragraph may contain a place and/or date of birth (list place, then date). Next, the author's educational background is listed. The degrees should be listed with type of degree in what field, which institution, city, state, and country, and year degree was earned. The author's major field of study should be lower-cased.

The second paragraph uses the pronoun of the person (he or she) and not the author's last name. It lists military and work experience, including summer and fellowship jobs. Job titles are capitalized. The current job must have a location; previous positions may be listed without one. Information concerning previous publications may be included. Try not to list more than three books or published articles. The format for listing publishers of a book within the biography is: title of book (city, state: publisher name, year) similar to a reference. Current and previous research interests end the paragraph.

The third paragraph begins with the author's title and last name (e.g., Dr. Smith, Prof. Jones, Mr. Kajor, Ms. Hunter). List any memberships in professional societies other than the IEEE. Finally, list any awards and work for IEEE committees and publications. If a photograph is provided, the biography will be indented around it. The photograph is placed at the top left of the biography. Personal hobbies will be deleted from the biography.