INTERNATIONAL JOURNAL OF NETWORK MANAGEMENT
Int. J. Network Mgmt 0000; **00**:1–16
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/nem

DNS-Class: Immediate classification of IP flows using DNS

Paweł Foremski[1]*, Christian Callegari[2] and Michele Pagano[2]

[1]*The Institute of Theoretical and Applied Informatics of the Polish Academy of Sciences, Gliwice, Poland*
[2] *Department of Information Engineering, University of Pisa, Italy*

SUMMARY

In the last years, we have witnessed a tremendous growth of the Internet, especially in terms of the amount of data being transmitted through the networks and new protocols being implemented. This poses a challenge for network administrators, who need adequate traffic classification tools for network management, e.g. to implement Quality of Service (QoS) requirements. In this paper, we employ real traffic traces to assess the usefulness of Domain Name System (DNS) information for traffic classification. We show that by inspecting DNS packets, it is possible to immediately classify a highly significant portion of the traffic. We present DNS-Class: an innovative, fast, and reliable flow-based classifier that on average yields 99.2% of True Positives with $<0.1\%$ of False Positives. We argue that DNS-Class represents an important development in the field of traffic classification, and that it can be a major element of a modular traffic classification system.

Received . . .

KEY WORDS: Network Management, IP Networks, Traffic Classification

1. INTRODUCTION

Internet traffic *classification*—or *identification*—is the act of matching IP packets to the applications that generated them. It is important for network infrastructures that implement Quality of Service (QoS) requirements. Traffic classifiers are the constitutive building blocks of e.g. traffic shapers, packet filters, and network monitors. For example, if VoIP traffic needs to be prioritized in a network, a traffic classifier would identify the IP packets with voice data—thus enabling the network administrator to route or queue them according to a special set of rules.

In a widely-known survey on traffic classification [1], the authors trace the beginnings of this field back to the year 1994, although first major publications appeared a decade later. In the past, a popular approach to identifying traffic was to examine the transport protocol port number, but the advent of Peer-to-Peer (P2P) traffic rendered this technique generally inadequate [2]. Nowadays, the two popular methods are Deep Packet Inspection (DPI) (pattern matching on packet payloads) and statistical classification (Machine Learning (ML) on flow features), but they have some limitations.

*Correspondence to: Paweł Foremski, IITiS PAN, ul. Bałtycka 5, 44-100 Gliwice, Poland. E-mail: pjf@iitis.pl

For example, statistical approaches need several IP packets to compute the flow features, whereas DPI needs access to packet payloads and generally cannot classify encrypted flows.

The goal of this work is to show that there is a significant portion of traffic passing through Internet gateways—namely, flows with domain names—that can be immediately classified, simply by looking at the first packet header and at the domain name. Moreover, by reliably sieving these flows out, our algorithm leaves the traffic that mostly is either P2P or Gaming. We also show interesting dependencies between IP flows and DNS traffic, demonstrating an innovative perspective for flow-based traffic analysis.

In this paper, we present DNS-Class: a novel method for classifying IP flows by inspecting DNS traffic transmitted in a computer network. First, our algorithm passively collects the information in DNS packets to find the domain names behind connection peers (e.g. `www.facebook.com`). Second, by running text classification on these names, DNS-Class reveals the generating application (e.g. HTTP). Combining this information with port numbers can further improve the classification performance, even though we consider that port numbers are unreliable.

The novelty of our method is explained in the fact we believe this is the first classifier employing DNS domain names as traffic features. One of the main strengths of DNS-Class is its ability to correctly classify an IP flow by only inspecting its first packet. We argue that this represents a significant step ahead with respect to the state of the art methods, which require several packets before being able to perform any classification.

The remainder of this paper is organized as follows. In Section 2 we comment on related works and refer the reader to required theoretical background. In Section 3 we present our DNS-Class algorithm. In Section 4 we analyze our traffic datasets and in Section 5 we use them to experimentally evaluate the algorithm. Finally, we discuss the results in Section 6 and give our conclusions in Section 7.

2. RELATED WORKS

To the best of our knowledge, there are no previous works on classifying Internet traffic using DNS, in a spirit that would be similar to e.g. statistical classification of flows.

In [3] however, D. Plonka and P. Barford present a system similar to DNS-Class, but with a different goal of profiling traffic and hosts. The algorithm matches domain names to flows and lets for two kinds of analysis: according to the presence of flow name (named/unnamed) and according to the domain name (hierarchical analysis). The system also uses the domain names to attribute hosts to 3 categories of “Torrent”, “Talk”, and “Game”. The authors give an analysis of cross-dependence between DNS and IP traffic, for two different networks. Comparing with our work, the paper adopts a vague set of traffic classes, which does not fit real applications or protocols. There is no ground-truth in the datasets used for experiments, hence the results were not validated. The authors do not employ an ML approach, and instead use a static set of hand-made patterns to match domain names with traffic profiles, which poses an operational limitation.

In [4], I. Bermudez et al. present “DN-Hunter”, a traffic analysis system exploiting the same idea of assigning DNS domain names to traffic flows as in DNS-Class. The authors give a detailed description of their DNS sniffer, and an analysis of DNS traffic limited to the scope of DN-Hunter.

Prepared using nemauth.cls
DOI: 10.1002/nem

WWW	NTP	Jabber	BitTorrent
nk.pl:80/TCP	pool.ntp.org:123/UDP	talk.google.com:5222/TCP	exodus.desync.com:6969/TCP
photos.nasza-klasa.pl:80/TCP	clock.fmt.he.net:123/UDP	talkx.l.google.com:5222/TCP	tracker.openbittorrent.com:80/UDP
www.playmobile.pl:80/TCP	ntp1.dlink.com:123/UDP	chat.facebook.com:5222/TCP	router.utorrent.com:6881/UDP
gg.hit.gemius.pl:80/TCP	time.nist.gov:123/UDP	xmpp.nktalk.pl:5222/TCP	tracker.publicbt.com:80/UDP
www.facebook.com:80/TCP	time-a.netgear.com:123/UDP	talk.l.google.com:5222/TCP	fr33dom.h33t.com:3310/TCP

Table I. Examples of input to Flow Classification stage: the top-5 flow names, with port numbers and transport protocol names. Polish domains are specific to the dataset. The BITTORRENT protocol uses dynamic port numbers, but its domain names can reveal the generating application.

However, the paper is devoted to uncovering the global Content Delivery Network (CDN) structure and describing related network phenomena—the DNS method is only a tool, which is neither used nor evaluated as a traffic classification method. Apart of interesting insights on the CDN industry, the authors present a technique that automatically extracts keywords from domain names related to transport protocol port numbers (sections 4.3 and 5.5 in that paper). This roughly corresponds to our Flow Classification described in Section 3.2, but the output is ambiguous, i.e. produces many keywords.

In a 2013 paper [5], P. Fiadino et al. present "HTTPTag": an on-line HTTP classification system, able to identify web-based applications and services. HTTPTag runs pattern matching on domain names extracted from HTTP headers, using a set of hand-made regular expressions. The authors claim that, using 380 regular expressions corresponding to 280 services, they are able to "classify more than 70% of the overall HTTP traffic volume caused by more than 88% of the web users in an operational 3G network". While their work has an alternative source of features and different goals than DNS-Class, it introduces the significance of domain names in classifying network traffic. Our work proposes a system that is more general (e.g. which applies to HTTPS too) and learns from data, without the need of manual training.

We do not present details on the theoretical background required to understand this paper. Instead, we refer the reader to [6] for foundations on ML, particularly to Chapters 1, 5.11, and 9.6. For a practical introduction to text classification, see Chapter 6 in [7]. For necessary background on DNS, see Chapter 7.1 in [8]. For further introduction to traffic classification, see [1], [9], or [10] for recent works.

3. THE DNS-CLASS ALGORITHM

The DNS-Class algorithm operates on traffic flows and consists of two stages: DNS Search and Flow Classification. The first stage assigns domain names to flows; the second runs text classification on domain names extended with port numbers and transport protocol names. See Table I for examples of input data. Below we give the taxonomy used throughout this paper.

We define as *flow* the set of packets belonging to the same connection neglecting their direction, i.e. packets having the same 5-tuple of $\langle IP_{src}, Port_{src}, IP_{dst}, Port_{dst}, Proto \rangle$, with the source src and destination dst optionally swapped. The IP, $Port$, and $Proto$ terms stand for IPv4 address, transport protocol port number, and transport protocol name, respectively.

Prepared using nemauth.cls
DOI: 10.1002/nem

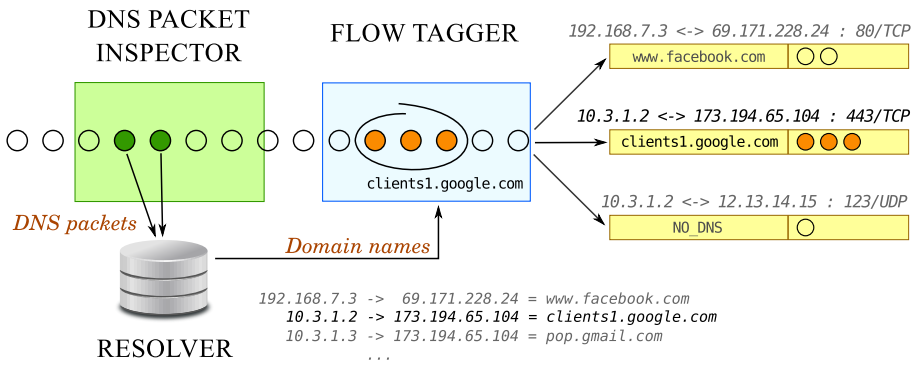

Figure 1. The DNS Search algorithm. Incoming packets are inspected for DNS information, grouped into flows, and flows are tagged with names. The algorithm discovered that the client `10.3.1.2` received earlier a DNS reply for the server `173.194.65.104` with the name of `clients1.google.com`, so it tagged the matching TCP connection with this domain name. The RESOLVER database holds names for client-server pairs; the same server address can appear under different names, depending on the client.

We also introduce an important notion of *named flows*: the flows for which the peer sending the first packet queried the DNS system. In such case, the remote host name is the *flow name*. On the contrary, if querying the DNS was not needed, then the flow is an *anonymous flow*. Note that querying DNS does not always involve physical communication with a DNS server, due to local caching of DNS information.

In the context of traffic classification, we employ the term *application*—with synonyms of *protocol* and *class*—in a broad meaning: a group of one or more computer programs (protocols) that generate (transport) Internet traffic. The information on the real application that generated a particular flow is called *the ground-truth*.

3.1. DNS Search

The architectural view of DNS Search is depicted in Figure 1. Its first element, DNS PACKET INSPECTOR, examines packets in DNS flows and updates the RESOLVER database with information on domain names for client-server pairs. The FLOW TAGGER queries this database each time a new flow starts and assigns DNS names, if possible. As the result of DNS Search, a set of flows is obtained, in which some elements have a domain name assigned.

The DNS Search algorithm exploits the fact that Internet connections are often anticipated by DNS query and response packets (which is shown in Section 4). For example, a web browser will resolve `clients1.google.com` to `173.194.65.104` before attempting a TCP connection to the corresponding web server. By intercepting the DNS response packet, an intermediary router can uncover the original domain name entered by the user into the web browser location bar.

In greater detail, DNS PACKET INSPECTOR dissects packets according to RFC1035 [11]. At its input, it takes successful replies to queries of `A` and `MX` types. The destination address in the IP header becomes *client address* ($Client_{addr}$: the host that asked for the domain name), and each address found in the DNS Answer becomes *server address* ($Server_{addr}$: the host that the client can connect to). The first element transmitted in the DNS Questions list is the domain name; it is stored in the RESOLVER database under an index of $\langle Client_{addr}, Server_{addr} \rangle$, for each $Server_{addr}$. This information will be cached for the next 10 hours of traffic—unless updated earlier by another

DOI: 10.1002/nem

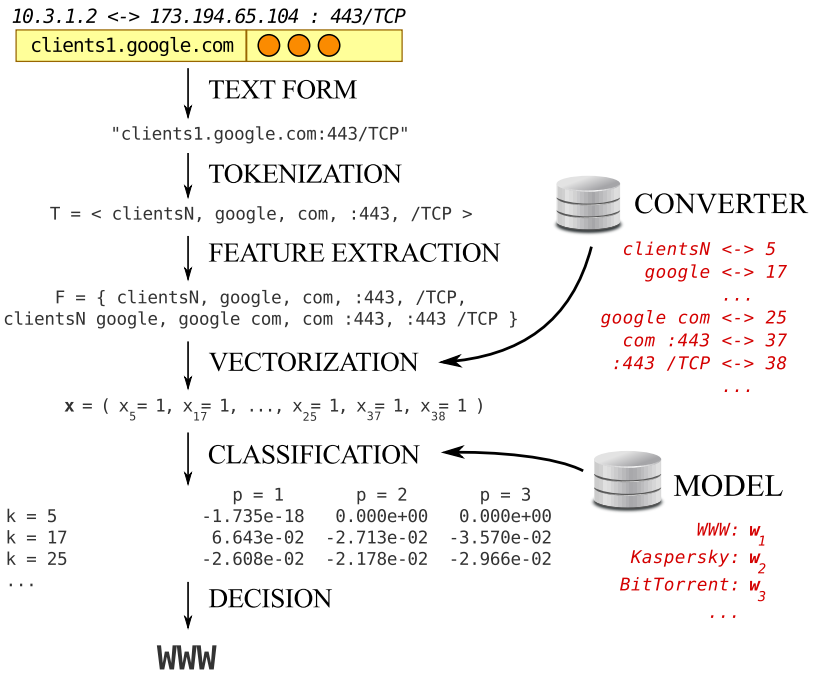

Figure 2. The DNS-Class algorithm. The input flow data is rewritten, converted into Vector Space Model $\mathbf{x}$, and classified using support vector classification. The CONVERTER holds 1-1 associations between textual features v_w and their integer counterparts v_k. The MODEL is constructed during system training and keeps weight vectors for each protocol.

DNS packet with the same RESOLVER index—which roughly imitates the caching mechanism usually found in the DNS resolvers running on the client machines. We chose the value of 10 hours arbitrarily; smaller values should work as well (see Section 6 in [4]).

The FLOW TAGGER is run for every TCP and UDP flow, but only on the arrival of the first packet. It queries the RESOLVER database for the domain name under $\langle IP_{src}, IP_{dst} \rangle$ index and assigns it to the flow (for the entire flow lifetime). If this fails, it tries the opposite direction, i.e. $\langle IP_{dst}, IP_{src} \rangle$. If this fails too, no name is attached to the flow and it is left anonymous, depicted as `NO_DNS` in Figure 1. Note that implementing a similar mechanism using reverse DNS lookup would fail: in [4], the authors proved experimentally that issuing a reverse lookup on the server IP of a randomly chosen named flow either returns a different domain name (62% of flows), or yields no answer at all (29% of flows).

The DNS Search algorithm is designed to be implemented on single machine, e.g. on a gateway router. However, the DNS PACKET INSPECTOR and FLOW TAGGER can be decoupled and implemented on two separate machines—e.g. intercepting DNS information on a local recursive DNS server, and tagging flows on a router—with small modifications to the algorithm.

3.2. Flow Classification

As the input to the classification stage of DNS-Class, we take the set of named flows. Figure 2 presents the classification of an exemplary flow: a TCP connection between `10.3.1.2` and `173.194.65.104` on port 443, having a domain name of `clients1.google.com` assigned.

DOI: 10.1002/nem

First, flow information is transformed into a textual form by concatenating the flow name, port number, and transport protocol name (using separating characters). DNS-Class can also operate without flow features—i.e. can classify solely by the domain name—which is demonstrated in section 5.2 pt 2. For the port number, we use the destination port of the first packet in a flow.

The text form is split by dots and dashes into a tuple of tokens T (T is limited to the last 6 tokens by default). Digits are replaced with the capital "N" character, except for the port number. The goal of this step is extracting keywords from domain names. We also tried word segmentation in long domain names as described in [12], but without meaningful effect on the overall classification performance.

The tuple of tokens T is converted into a set of text features F, by extracting all word unigrams and bigrams. The algorithm has a CONVERTER database, which is used as an invertible function C that maps these unigrams and bigrams to integers, as shown in Equations 1 and 2:

$$C(v_w) = v_k, \tag{1}$$

$$C^{-1}(v_k) = v_w, \tag{2}$$

where $v_w \in W$, the set of all known unigrams and bigrams, and $v_k \in K$, $K = \{1, 2, \ldots, n\}$. Note that $|W| = |K| = n$. The set W is constructed during the training of DNS-Class and held constant during classification, and so is K. In case a given unigram or bigram v_w cannot be found in the database, the corresponding v_k is 0, i.e.

$$C(v_w) = 0 \quad v_w \notin W, \tag{3}$$

which means that the text feature v_w is dropped and not considered in next algorithm steps.

In the VECTORIZATION step, the set of text features F is converted into a sparse feature vector $\mathbf{x}$ of size n, in which each dimension corresponds to a text feature. In other words, a Vector Space Model (VSM) representation of the text input is constructed [13]: $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, where

$$x_j = \begin{cases} 1 & j \in F' \\ 0 & j \notin F' \end{cases} \qquad \forall j \in K, \tag{4}$$

and F' is the set of text features F converted to integers using function C. We also tried another values for x_j—including the popular tf-idf metric [13]—but we experienced no major impact on the overall system performance.

For the CLASSIFICATION step, we employ direct multi-class support vector classification by Crammer and Singer [14] with a linear kernel, as implemented in LIBLINEAR [15, 16] and LIBSHORTTEXT [17]. In this step, the MODEL database is queried for weight vectors $\mathbf{w}_p$, for each target class p:

$$\mathbf{w}_p = (w_p^{(1)}, w_p^{(2)}, \ldots, w_p^{(n)}), \tag{5}$$

DOI: 10.1002/nem

where each $w_p^{(j)}$ term corresponds to the weight of text feature $j \in K$, and $p \in P$, $P = \{1, 2, \ldots, m\}$: the set of all network protocols to be recognized by the classification system. Let

$$\alpha(\mathbf{x}, p) = \mathbf{w}_p^T \mathbf{x}, \tag{6}$$

be the *decision value* for protocol p given feature vector $\mathbf{x}$, then the *decision function* is

$$\arg\max_p \ \alpha(\mathbf{x}, p), \tag{7}$$

which predicts the protocol behind $\mathbf{x}$. In other words, in the CLASSIFICATION step, we search for the protocol p that maximizes Equation 6 for given feature vector $\mathbf{x}$.

During training of DNS-Class, weight vectors $\mathbf{w}_p$ are initialized using training instances of text, according to Equations 8 and 9:

$$\begin{array}{ll} \min\limits_{\{\mathbf{w}_p\},\{\xi_i\}} & \frac{1}{2}\sum_p \|\mathbf{w}_p\|^2 + C\ \sum_i \xi_i \\ \text{s.t.} & \alpha(\mathbf{x}_i, y_i) - \alpha(\mathbf{x}_i, p) \geq \gamma_i(p) - \xi_i \quad \forall p, i \end{array} \tag{8}$$

$$\gamma_i(p) = \begin{cases} 0 & y_i = p \\ 1 & y_i \neq p \end{cases}, \tag{9}$$

where i is the number of text instance, $C \in \mathbb{R}_{>0}$ is the regularization parameter, $\xi_i \in \mathbb{R}_{\geq 0}$ are slack variables, and $y_i \in P$ is the true protocol behind feature vector $\mathbf{x}_i$—that is, the ground-truth label. Roughly speaking, the goal of the optimization described by Equation 8 is to have high $w_p^{(j)}$ values for features that are specific to protocol p, and low values for features that are common for all protocols. We refer the reader to the cited papers for further details on our classification algorithm, especially to [15] and [17].

In the last step (DECISION), the protocol p selected according to Equation 7 is translated into corresponding textual representation. For example, in Figure 2, $p = 1$ stands for `WWW`.

3.3. Rationale

DNS-Class is a specialized traffic classifier that targets named flows and DNS traffic passing through Internet gateways. This usually corresponds to a significant portion of the whole traffic, as specified in [4], where Bermudez et al. claim that for HTTP and TLS flows, the portion of named flows usually exceeds 90% in most of their highly representative datasets. Given that HTTP is nowadays considered to be the dominant protocol in residential customer traffic [18], the actual portion of named flows transmitted through ordinary Internet gateways can be even higher than what our study shows.

In Figure 3, we present one of possible scenarios for DNS-Class: traffic traveling through a gateway is classified in a modular system. Each module is responsible for handling only one part of the traffic, according to several criteria. If an input flow cannot be classified, it is handed over to the next module in the "chain" of classifiers. In such scenario, the goal of DNS-Class is classifying only the named flows and DNS, leaving the anonymous flows for other modules. For example, DNS-Class can be augmented with statistical classifiers, fine-grained methods [19], or even with other specialized classifiers like Skype-Hunter [20]. Note that our work introduces a reliable criterion for

Prepared using nemauth.cls
DOI: 10.1002/nem

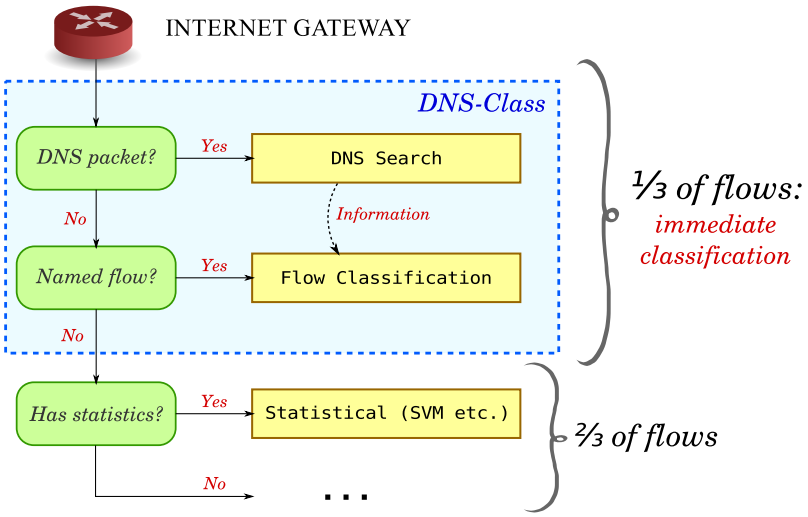

Figure 3. Modular traffic classifier. We propose an algorithm that targets one-third of network traffic in the investigated ISP network. DNS-Class immediately classifies named flows, leaving anonymous flows for other classifiers.

distinguishing traffic flows that DNS-Class can classify: the presence of a domain name attached to the flow.

DNS-Class has interesting advantages over existing methods. In particular, our algorithm immediately classifies network flows, requiring just the IP header of the first packet and the information extracted from DNS query-response conversations. Comparing with statistical traffic classifiers, our algorithm is quicker, because it does not wait until enough flow data is collected to compute the statistics. Comparing with DPI, DNS-Class does not inspect the payload of the packets (except for DNS packets), which makes it resistant to Transport Layer Security (TLS) encryption. We thus believe that DNS-Class represents a novel and important development in the field of traffic classification.

4. DATASETS AND TRAFFIC ANALYSIS

In this section, we analyze the traffic datasets that we used for validating DNS-Class, and which will be used in the next section for presenting the practice of applying DNS-Class to real network traffic. We also share our findings on how Internet protocols depend on DNS.

4.1. Traffic traces

We collected the network traffic during May-June 2012 and January 2013 at a Polish ISP company in Upper Silesia that serves residential customers. In both cases, packet capture was run for around one week on the same Point-to-Point over Ethernet (PPPoE) server that handled a few hundred users. The maximum amount of captured packet data was limited due to storage constraints; the Ethernet and PPPoE headers were removed too. Table II summarizes the datasets, and Figure 4 presents one exemplary day of traffic in the ASNET1 dataset.

DOI: 10.1002/nem

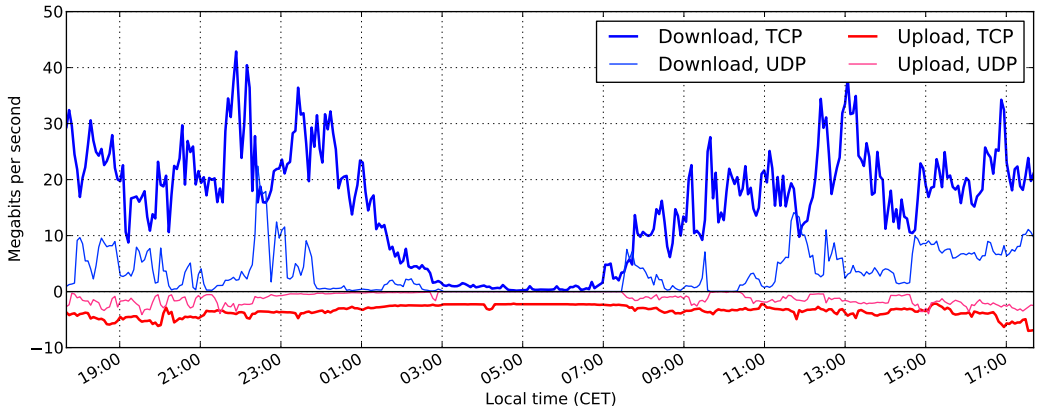

Figure 4. One day of traffic in ASNET1. Total network bandwidth usage of client downloads and uploads, for TCP and UDP. The data were collected June 1-2, 2012 and represents 4-minute averages.

Dataset	Start	Duration	Src. IP	Dst. IP	Packets	Bytes	Avg. Util	Avg. Flows (/5 min.)	Payload
Asnet1	2012-05-26 17:40	216h	1,828 K	1,530 K	2,525 M	1,633 G	18.0 Mbps	7.7 K	92 B
Asnet2	2013-01-24 16:26	168h	2,503 K	2,846 K	2,766 M	1,812 G	25.7 Mbps	12.0 K	84 B

Table II. Datasets used for experimental validation. The ASNET2 dataset was captured 8 months after ASNET1, with smaller packet size limit and shorter capture duration, but it contains more traffic. Both datasets contain real traffic of the same population of a few hundred residential ISP customers.

We established the ground-truth using DPI, as implemented in the LIBPROTOIDENT library version 2.0.7[†], published by the University of Waikato [21]. We embedded this library in our software toolkit FLOWCALC, which converts PCAP files into flow-level summaries in the ARFF file format[‡]. We made several corrections to the DPI results by analyzing the traffic traces manually, according to our knowledge. For instance, we noticed several flows on ports 6969, 2710, and 3310 being erroneously classified as HTTP_NONSTANDARD instead of BITTORRENT; another problem was an over-matching rule for the TEREDO protocol.

We adopted definitions of traffic classes from LIBPROTOIDENT, ignoring the transport protocol name and encryption level: e.g. KASPERSKY instead of KASPERSKY_TCP and KASPERSKY_UDP, and WWW instead of HTTP and HTTPS. We also used the MAIL class as an aggregate for POP3, SMTP, and IMAP. For our experiments with the CAIDA port-based classifier in Section 5.2 pt 1, we translated the traffic classes used by that classifier.

Next, we sanitized the datasets by removing incomplete TCP sessions, and by dropping the traffic that is specific for Local Area Network (LAN) environments—e.g. DHCP, NETBIOS, and SSDP. Finally, we ran our DNS Search algorithm described in Section 3.1 to discover the domain names of the network flows in our datasets.

[†]Subversion revision number 154
[‡]See `http://mutrics.iitis.pl/flowcalc`

DOI: 10.1002/nem

4.2. Traffic characteristics

In order to show how different applications depend on DNS, we divide the set of all network protocols into three groups: 1) traditional client-server protocols (e.g. browsing, e-mail, streaming), 2) P2P and Gaming traffic, and 3) other. The last group consists of DNS traffic and the flows for which our ground-truth method failed. Table III (pp. 17) presents results of traffic analysis, and is the basis for this section. For the sake of brevity, we report only on the ASNET1 dataset, leaving the ASNET2 dataset for temporal stability evaluation in Section 5.2 pt 5. For examples of flow names and port numbers, see Table I.

For validating DNS-Class, we need flows with both the ground-truth label and the domain name—this was the case for 26% of flows in ASNET1. DNS-Class also identifies DNS packets directly in the DNS Search algorithm, so in total our algorithm targets 37% of all flows in ASNET1. As can be seen in Table IIIa, network protocols differ in how much they depend on DNS. In next sections we will only consider the protocols for which at least 10% of flows have a domain name (in order to have enough training data), except for BITTORRENT and SKYPE, which were included for their popularity in the traffic classification literature.

For the ASNET1 dataset, we found that:

1. *27% of flows have a domain name.* We believe the real portion of named flows is higher, because our DNS Search algorithm is imperfect and the ASNET1 dataset has limited amount of packet payload.
   (a) Flows of traditional client-server protocols vary in their dependence on DNS, but generally this class of flows often incurs DNS queries: 78% of traditional flows have a domain name. Comparing to the work by Bermudez et al. [4], this is smaller—but similar (see Table 2 in that paper).
   (b) P2P applications and computer games almost never employ DNS for communication: on average, only 0.2% of their flows have a domain name. These protocols do not need DNS for communication between peers. For example, BitTorrent trackers point to seeders and leechers by their IP addresses; similarly, game servers also list the players by IP addresses, and the exchange of game information occurs directly between the peers.
2. *50% of bytes and 44% of packets travel in named flows*. The average size of named flows is two times higher than the size of anonymous flows (200 KB vs. 110 KB).
   (a) For traditional protocols, 74% of bytes and 73% of packets are transmitted in named flows.
   (b) For P2P and Gaming traffic, this is 0.0018% and 0.02% for bytes and packets, respectively.
3. *If a flow has a domain name, it is almost certainly not P2P nor Gaming.* Only 0.4% of named flows are P2P or games, which confirms results of Plonka and Barford [3].
   (a) This phenomenon can be practically applied as a quick method for ensuring that a flow does not belong to a P2P application or a computer game.
   (b) Conversely, if a flow does *not* have a domain name—and at the same time it is not a DNS flow—then it is either P2P or Gaming with a probability of $>77\%$.

DOI: 10.1002/nem

5. EXPERIMENTAL EVALUATION

In this section, we present the practice of using DNS-Class in a real network, in different setups. We also compare DNS-Class to an established traffic classification method.

The experiments presented in this section were designed to evaluate the robustness of DNS-Class and to assert that combining domain names with port numbers is meaningful. The results are presented in Table IV (pp. 18), which is the main table for this section.

5.1. Methodology

For each experiment with DNS-Class, we begin by tuning the algorithm parameters according to the experiment description. Then we evaluate the classification performance and robustness: in experiments 2-4 we employ 10-fold cross-validation on ASNET1 [6], and in experiment 5 we train on ASNET1 and test on ASNET2.

For given protocol p, we measure the classification performance according to two complementary metrics of True Positives ($\%TP_p$) and False Positives ($\%FP_p$):

$$\%TP_p = \frac{|TP_p|}{|F_p|} \cdot 100\%, \quad \%FP_p = \frac{|FP_p|}{|F'_p|} \cdot 100\%, \tag{10}$$

where TP_p is the set of true positives for protocol p, F_p is the set of testing flows that belong to p, FP_p is the set of false positives for p, and F'_p is the set of testing flows that do not belong to p. The set of *true positives* for protocol p consists of the flows that were properly attributed to p during classification. On the contrary, FP_p consists of the flows improperly attributed to p. The *testing flows* are the flows from the dataset that were used for testing the classifier.

For measuring the overall classification performance, we simply adopt the average for all protocols:

$$\%TP = \frac{\sum_p \%TP_p}{|P|}, \quad \%FP = \frac{\sum_p \%FP_p}{|P|}. \tag{11}$$

Another option would be to compute the weighted average according to the number of flows in each class. However, the result would be heavily biased towards the WWW class, which holds the vast majority of flows.

5.2. Experiments

1) Traditional port number classifier (Figures 5 and 6): We begun our experiments by evaluating a traditional port number classifier on our datasets. We chose the CAIDA Coral Reef suite version 3.9.1 [22] as an established port-based classifier. Because this classifier does not need training, we used all flows from the ASNET1 dataset for testing.

First, we evaluated the classifier on all protocols in groups (1) and (2) defined in Section 4. The algorithm exhibited very good results for some traffic classes, but in most cases it failed: the %TP metric was 33%, as visible in Figure 5. The figure also presents the %TP metric for packets and bytes, computed similarly. In order to enable direct comparison with DNS-Class, we also evaluated

DOI: 10.1002/nem

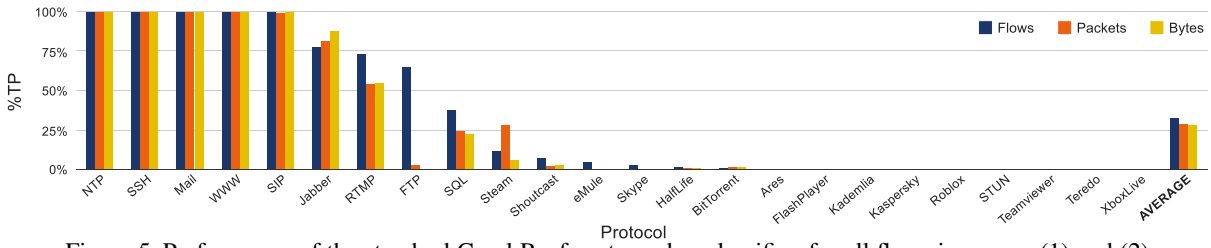

Figure 5. Performance of the standard Coral Reef port number classifier, for all flows in groups (1) and (2) (see Table III). The %TP metric is 33%, 29%, and 28%—for flows, packets, and bytes, respectively.

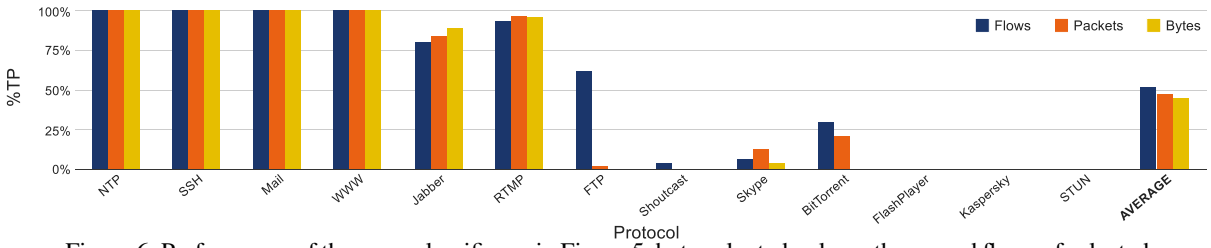

Figure 6. Performance of the same classifier as in Figure 5, but evaluated only on the named flows of selected protocols. %TP is 52%, 47%, and 45%—for flows, packets, and bytes, respectively.

the port classifier on our target traffic, i.e. on named flows. The performance improved, but still was quite low: %TP of 52%, as visible in Figure 6.

2) Classification by domain name (Table IVa): In experiment 2, we run the algorithm on sole DNS domain features, i.e. we classified the traffic just by the domain name, neglecting the transport protocol and the port number. Using the options of our software, we forced the algorithm to ignore the last two tokens in the input data (see Section 3.2). We trained and tested the system using the ASNET1 dataset. Results are presented in Table IVa, with %TP and %FP of 82% and 1%, respectively.

The three most common errors made by the algorithm were: classifying `poczta.interia.pl:995/TCP` as WWW (instead of MAIL), classifying `tracker.openbittorrent.com:80/UDP` as WWW (instead of BITTORRENT), and classifying `talkx.l.google.com:443/TCP` as JABBER (instead of WWW).

3) Classification by port number (Table IVb): Conversely, in experiment 3, classification relied only on the port number and on the transport protocol name, i.e. we ignored the domain name. Again, using the software options of DNS-Class, we skipped all the tokens but the last two. Using the same dataset for training and testing, we obtained the results presented in Table IVb, with %TP and %FP of 92% and 2%, respectively.

For this experiment, the most common errors were connected with the KASPERSKY protocol, e.g. erroneously classifying `ksn2-12.kaspersky-labs.com:443/TCP` as WWW.

4) Classification by domain name and port number (Table IVc): Finally, in experiment 4 we evaluated the full DNS-Class algorithm, i.e. classification by domain name, port number, and transport protocol name. This is the main experiment that presents full capabilities of our algorithm. Again, we used the ASNET1 dataset for training and testing, but we run DNS-Class on full textual representations of flows, exemplified in Table I. On average, we obtained %TP of 99% and %FP of $<0.1\%$, which is displayed in greater detail in Table IVc.

DOI: 10.1002/nem

The most common source of errors were due to the SHOUTCAST and RTMP protocols. Textual representations of their flows—for instance, `petelo.streams.bassdrive.com:80/TCP` and `brightcove-32.fcod.llnwd.net:443/TCP`—were incorrectly classified as WWW. Interestingly, we also found errors in the ground-truth: e.g. textual inputs like `www.facebook.com:80/TCP`, `imppl.tradedoubler.com:80/TCP`, and `plus.google.com:80/TCP` were incorrectly attributed to the BITTORRENT protocol instead of WWW.

5) Temporal stability (Table IVd): In experiment 5, we evaluated the temporal stability of DNS-Class, i.e. whether a classification model can be used in the same network after a longer period of time. We created the model using ASNET1, but for testing we took the ASNET2 dataset, collected after 8 months. We obtained %TP and %FP of 98% and $<0.1\%$, respectively. Detailed results are given in Table IVd.

Majority of errors were due to the FLASHPLAYER protocol: DNS-Class failed for inputs like `telegraph.justin.tv:443/TCP` and `qfpgameservermad05.miniclip.com:4000/TCP`, incorrectly classifying them as `WWW`. Again, we noticed some errors in the ground-truth labels, e.g. `php.hdcmct.com:2710/TCP` attributed to WWW instead of BITTORRENT, and `www.facebook.com:443/TCP` attributed to BITTORRENT instead of WWW.

6. DISCUSSION

Our experiments with DNS-Class showed that:

1. *Traditional port number classifier is unreliable for the traffic targeted by DNS-Class.* We evaluated CoralReef port number classifier on named flows, and the experiment showed it could not replace DNS-Class, due to poor performance (%TP of 52%).
   (a) Classifying only by port number is unreliable also in the general case of all flows (i.e. named and anonymous flows). This was demonstrated in several papers on traffic classification (e.g. [2]), and in our experiment 1 (see Figure 5). The classifier worked properly for traditional protocols (e.g. NTP), but failed for newer P2P protocols (e.g. BITTORRENT).
   (b) The %TP metric for port number classification was better in the case of narrowing the traffic to named flows only (Figure 6). For example, %TP improved for RTMP and BITTORRENT. This represents an increase in performance at the cost of smaller traffic scope.
2. *It is possible to classify some protocols using domain names only.* Experiment 2 (Table IVa) demonstrated that in 7 out of 13 traffic classes, it was possible to identify the protocol behind network flows, with %TP above 90%.
   (a) The WWW class collected majority of wrong classifications (%FP of 14.3%). This is due to popularity of this class, and because the set of words in website domains stands for a huge portion of tokens that DNS-Class can find in domain names of other protocols.
   (b) We noticed poor performance for FTP and FLASHPLAYER classes (%TP below 50%), and moderate results for JABBER, MAIL, RTMP, and BITTORRENT (%TP below 90%). Probably, domain names of these protocols contain small number of tokens that could distinguish them from the WWW class.

Prepared using nemauth.cls
DOI: 10.1002/nem

   (c) There exist domains that are used for delivering more than one service at the same time—for example, `poczta.interia.pl`: an e-mail service delivered through traditional SMTP/POP3 protocols, and through a web interface over HTTPS.

3. *Port number is an important traffic feature for named flows.* Experiment 3 (Table IVb) proved that in DNS-Class, port numbers can be used for successful classification in many cases, with %TP and %FP of 92.1% and 1.5%, respectively.
   (a) DNS-Class employs a machine-learning algorithm—instead of relying on a static database, as in the CoralReef port number classifier—hence it learns the port numbers from the traffic in the dataset. The system was also able to properly classify named flows of P2P protocols: BITTORRENT (%TP of 99.3%) and SKYPE (%TP of 94.6%), with no false positives.
   (b) Again, the WWW class collected majority of errors (%FP of 19.0%), but this time due to the fact that several other protocols use the port numbers of 80 and 443 (traditionally linked with HTTP and HTTPS), for example: KASPERSKY, FLASHPLAYER, and SHOUCAST.
4. *Full DNS-Class algorithm is reliable.* In experiment 4 (Table IVc), DNS-Class obtained %TP of 99.2% and %FP of $<0.1\%$. This result proves our argument that it is possible to classify a significant portion of Internet flows using just the first packet and the flow name.
   (a) Combining port numbers with domain names improved the classification performance for each evaluated protocol.
   (b) The WWW class collected majority of errors due to the reasons already given in points 2a and 3b above, but with a much lower %FP of $<0.1\%$.
   (c) DNS-Class proved to be better than our ground-truth method in a few cases, as highlighted in Section 5.2, experiments 4 and 5.
5. *DNS-Class is stable over time.* In experiment 5 (Table IVd) we demonstrated that one can use the same model for classifying traffic after 8 months since training. Our algorithm achieved %TP of 97.7%, but still with %FP below 0.1%.
   (a) The overall system performance was worse than for a fresh model, but still acceptable. The performance was lower because several new services—and several new domain names—appeared on the Internet.

7. CONCLUSIONS

In this paper, we demonstrated a practical system that immediately classifies a highly significant portion of Internet traffic, using DNS information and first packets of IP flows. In Section 3, we described an algorithm for tagging flows with domain names, and an algorithm for classifying these flows. We analyzed the dependency of network protocols on DNS in Section 4, showing e.g. that 1) it is possible to find the domain name for 27% of flows, 50% of bytes, and 44% of packets in the investigated ISP network, and 2) that P2P and Gaming applications almost never employ DNS. Finally, in Section 5 we demonstrated the robustness of DNS-Class by evaluating it on two traces of real traffic, obtaining very good performance results.

Prepared using nemauth.cls
DOI: 10.1002/nem

A potential operational limitation of DNS-Class is obtaining the training data. We searched for DNS data using heuristics (Section 3.1), but more accurate methods can be employed, e.g. using a single application traffic sniffer to capture the DNS packets directly [23]. Another issue is unknown detection, i.e. detecting applications that were not present in the training data. We leave this for future work, with a preliminary idea to introduce thresholds on the number of non-zero elements in the feature vector $\mathbf{w}_p$, and on the value of the decision function (Eq. 7). Extending our work to IPv6 and other transport protocols should be straightforward.

We conclude that DNS is useful for traffic classification. We presented an important and innovative approach to IP flow analysis in the context of traffic classification. Our work can be a motivation to employ DNS information as a traffic feature and to enrich flow exporting formats like Netflow or IPFIX with flow domain names. Finally, in Section 3.3 we gave our vision for DNS-Class that explains its importance for the field of traffic classification.

ACKNOWLEDGEMENTS

This work was funded by the Polish National Science Centre, under research grant nr 2011/01/N/ST6/07202: project "Multilevel Traffic Classification", `http://mutrics.iitis.pl/`. The traffic traces were collected thanks to ASN Sp. z o.o., `http://www.asn.pl/`. We would like to thank Michał Gorawski and Michał Romaszewski of IITiS PAN for proofreading manuscripts of this paper.

REFERENCES

1. Nguyen TT, Armitage G. A survey of techniques for internet traffic classification using machine learning. *Communications Surveys & Tutorials, IEEE* 2008; **10**(4):56–76.
2. Karagiannis T, Broido A, Brownlee N, Claffy KC, Faloutsos M. Is P2P dying or just hiding? *Global Telecommunications Conference, 2004. GLOBECOM'04. IEEE*, vol. 3, IEEE, 2004; 1532–1538.
3. Plonka D, Barford P. Flexible Traffic and Host Profiling via DNS Rendezvous. *Workshop SATIN*, 2011.
4. Bermudez I, Mellia M, Munafò MM, Keralapura R, Nucci A. DNS to the Rescue: Discerning Content and Services in a Tangled Web. *Proceedings of the 12th ACM SIGCOMM Conference on Internet Measurement*, vol. 1101, 2012; 12.
5. Fiadino P, Bär A, Casas P. HTTPTag: A Flexible On-line HTTP Classification System for Operational 3G Networks. *International Conference on Computer Communications, 2013. INFOCOM'13. IEEE*, IEEE, 2013.
6. Duda RO, Hart PE, Stork DG. *Pattern classification*. John Wiley & Sons, 2012.
7. Bird S, Klein E, Loper E. *Natural language processing with Python*. O'Reilly, 2009.
8. Tanenbaum AS, Wetherall DJ. *Computer Networks*. 5 edn., Prentice Hall, 2010.
9. Callado A, Kamienski C, Szabó G, Gero B, Kelner J, Fernandes S, Sadok D. A survey on internet traffic identification. *Communications Surveys & Tutorials, IEEE* 2009; **11**(3):37–52.
10. Foremski P. On different ways to classify Internet traffic: a short review of selected publications. *Theoretical and Applied Informatics* 2013; **25**(2).
11. Mockapetris P. *RFC 1035 Domain Names - Implementation and Specification*. Internet Engineering Task Force November 1987. URL `http://tools.ietf.org/html/rfc1035`.
12. Norvig P. Word segmentation. *Beautiful data: the stories behind elegant data solutions*, Segaran T, Hammerbacher J (eds.). O'Reilly Media, Incorporated, 2009; 221–227. URL `http://norvig.com/ngrams/ch14.pdf`.
13. Salton G, Wong A, Yang CS. A vector space model for automatic indexing. *Communications of the ACM* 1975; **18**(11):613–620.
14. Crammer K, Singer Y. On the algorithmic implementation of multiclass kernel-based vector machines. *The Journal of Machine Learning Research* 2002; **2**:265–292.
15. Fan RE, Chang KW, Hsieh CJ, Wang XR, Lin CJ. LIBLINEAR: A library for large linear classification. *The Journal of Machine Learning Research* 2008; **9**:1871–1874.

Prepared using nemauth.cls
DOI: 10.1002/nem

16. Keerthi SS, Sundararajan S, Chang KW, Hsieh CJ, Lin CJ. A sequential dual method for large scale multi-class linear SVMs. *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2008; 408–416.
17. Yu HF, Ho CH, Juan YC, Lin CJ. Libshorttext: A library for short-text classification and analysis. *Technical Report*, National Taiwan University 2013. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/libshorttext.pdf`.
18. Maier G, Feldmann A, Paxson V, Allman M. On dominant characteristics of residential broadband internet traffic. *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, ACM, 2009; 90–102.
19. Park B, Won Y, Chung J, Kim Ms, Hong JWK. Fine-grained traffic classification based on functional separation. *International Journal of Network Management* 2013; .
20. Adami D, Callegari C, Giordano S, Pagano M, Pepe T. Skype-Hunter: A real-time system for the detection and classification of Skype traffic. *International Journal of Communication Systems* 2012; **25**(3):386–403.
21. Alcock S, Nelson R. Libprotoident: Traffic Classification Using Lightweight Packet Inspection. *Technical Report*, University of Waikato 2013. URL `http://www.wand.net.nz/publications/lpireport`.
22. Keys K, Moore D, Koga R, Lagache E, Tesch M, Claffy K. The architecture of CoralReef: an Internet traffic monitoring software suite. *PAM2001, Workshop on Passive and Active Measurements, RIPE*, Citeseer, 2001.
23. Foremski P. Tracedump: A Novel Single Application IP Packet Sniffer. *Theoretical and Applied Informatics* 2012; **24**(1):23–31.

DOI: 10.1002/nem

a)

	Protocol	All traffic			Traffic with domain name			Named flows	Selected?
		Flows	Packets (K)	Bytes (M)	Flows	Packets (K)	Bytes (M)		
	WWW	4,956,030	1,384,874	1,107,301	3,943,929	1,027,737	821,322	79.58%	yes
	Kaspersky	43,340	485	16	16,391	193	7.3	37.82%	yes
	NTP	34,160	102	4.7	10,241	47	2.1	29.98%	yes
	STUN	20,786	540	276	4,646	35	1.8	22.35%	yes
	Mail	20,293	8,590	5,558	14,391	7,950	5,455	70.92%	yes
	SIP	18,498	374	127	78	69	22	0.42%	-
	Teredo	12,504	923	59	0	0	0	0.00%	-
1)	SSH	9,424	1,348	837	973	1,017	799	10.32%	yes
	Jabber	4,752	558	62	4,530	539	61	95.33%	yes
	SQL	4,752	3,196	1,086	9	62	33	0.19%	-
	Teamviewer	1,185	1,683	231	72	13	4.3	6.08%	-
	FlashPlayer	1,039	142	48	329	10	2.0	31.67%	yes
	RTMP	821	18,073	13,330	373	3,283	2,193	45.43%	yes
	FTP	209	294	243	182	254	211	87.08%	yes
	Shoutcast	160	4,335	2,481	142	4,085	2,304	88.75%	yes
	BitTorrent	5,034,169	434,230	294,422	14,720	111	5.3	0.29%	yes
	Kademlia	1,222,167	4,273	284	0	0	0	0.00%	-
	eMule	290,101	98,558	71,112	0	0	0	0.00%	-
	Steam	179,910	2,872	955	88	10	0.9	0.05%	-
2)	Skype	172,171	66,861	26,221	236	1.5	0.10	0.14%	yes
	Ares	10,990	2,407	271	0	0	0	0.00%	-
	XboxLive	6,564	2,148	288	7	2.7	0.08	0.11%	-
	HalfLife	3,832	57	2.7	2	0.01	0.001	0.05%	-
	Roblox	364	25,595	5,297	0	0	0	0.00%	-
	DNS	1,817,572	9,620	581	0	0	0	0.00%	yes
3)	Unknown	1,717,714	374,586	136,876	209,127	28,867	8,651	12.17%	-
	Total:	15,583,507	2,446,723	1,667,969	4,220,466	1,074,286	841,076		

b)

Protocol group	Traffic with domain name		
	Flows	Packets	Bytes
Traditional (1)	77.93%	73.33%	73.56%
P2P & Games (2)	0.22%	0.02%	<0.01%
Other (3)	5.92%	7.51%	6.29%
All traffic:	27.08%	43.91%	50.42%

c)

Result of DNS search	Distribution of flows		
	Traditional (1)	P2P & G. (2)	Other (3)
Success	94.69%	0.36%	4.96%
Failure	9.96%	60.77%	29.27%
All traffic:	32.91%	43.18%	22.69%

Table III. Results of traffic analysis in the ASNET1 dataset. Table IIIa lists all significant protocols in each group—traditional client-server (1), P2P & Games (2), and Other (3)—and gives numerical values on the traffic and its dependency on DNS. The last column indicates which protocols were chosen for experimental validation. Table IIIb analyses how many flows, packets, and bytes in each group have a domain name. Table IIIc shows the distribution of flows among the groups, depending on the result of the DNS Search algorithm.

Prepared using nemauth.cls
DOI: 10.1002/nem

a)

	FPlayer	FTP	Jabber	Kasp.	Mail	NTP	RTMP	SSH	STUN	Shcast	Skype	Torrent	WWW	%TP	%FP
FPlayer	39.8%												60.2%	**39.8%**	<0.1%
FTP		45.6%											54.4%	45.6%	<0.1%
Jabber			66.5%										33.5%	66.5%	<0.1%
Kasp.				100.0%										100.0%	0.0%
Mail					79.7%								20.3%	79.7%	<0.1%
NTP						99.9%							0.1%	99.9%	<0.1%
RTMP							71.3%						28.7%	71.3%	<0.1%
SSH								99.9%					0.1%	99.9%	<0.1%
STUN									98.0%				2.0%	98.0%	<0.1%
Shcast										93.8%			6.2%	93.8%	<0.1%
Skype											99.2%		0.8%	99.2%	<0.1%
Torrent												68.1%	31.9%	68.1%	<0.1%
WWW	<0.1%	<0.1%	<0.1%		<0.1%	<0.1%	<0.1%	<0.1%	<0.1%	<0.1%	<0.1%	<0.1%	99.9%	99.9%	**14.3%**
													Average:	81.7%	1.1%

b)

	FPlayer	FTP	Jabber	Kasp.	Mail	NTP	RTMP	SSH	STUN	Shcast	Skype	Torrent	WWW	%TP	%FP
FPlayer	87.5%	<0.1%											12.5%	87.5%	0.0%
FTP		99.9%											<0.1%	99.9%	<0.1%
Jabber			100.0%											100.0%	0.0%
Kasp.				25.3%									74.7%	**25.3%**	0.0%
Mail		<0.1%			99.9%									99.9%	0.0%
NTP						100.0%								100.0%	0.0%
RTMP							96.3%						3.7%	96.3%	0.0%
SSH								100.0%						100.0%	0.0%
STUN		0.4%							99.6%					99.6%	<0.1%
Shcast		0.6%								94.2%			5.2%	94.2%	<0.1%
Skype									5.4%		94.6%			94.6%	0.0%
Torrent		<0.1%							0.1%			99.3%	0.5%	99.3%	0.0%
WWW		<0.1%								<0.1%			99.9%	99.9%	**19.0%**
													Average:	92.1%	1.5%

c)

	FPlayer	FTP	Jabber	Kasp.	Mail	NTP	RTMP	SSH	STUN	Shcast	Skype	Torrent	WWW	%TP	%FP
FPlayer	97.4%												2.6%	97.4%	<0.1%
FTP		99.9%											<0.1%	99.9%	0.0%
Jabber			100.0%											100.0%	0.0%
Kasp.				100.0%										100.0%	0.0%
Mail					100.0%									100.0%	<0.1%
NTP						100.0%								100.0%	0.0%
RTMP							98.7%						1.3%	98.7%	<0.1%
SSH								100.0%						100.0%	0.0%
STUN									100.0%					100.0%	<0.1%
Shcast										94.7%			5.3%	**94.7%**	<0.1%
Skype											99.2%	0.8%		99.2%	0.0%
Torrent									<0.1%			99.5%	0.5%	99.5%	<0.1%
WWW	<0.1%				<0.1%		<0.1%			<0.1%			99.9%	99.9%	**0.1%**
													Average:	99.2%	<0.1%

d)

	FPlayer	FTP	Jabber	Kasp.	Mail	NTP	RTMP	SSH	STUN	Shcast	Skype	Torrent	WWW	%TP	%FP
FPlayer	78.0%		3.6%		0.3%								18.1%	**78.0%**	<0.1%
FTP		96.2%											3.8%	96.2%	<0.1%
Jabber			100.0%											100.0%	<0.1%
Kasp.				100.0%										100.0%	0.0%
Mail					100.0%									100.0%	<0.1%
NTP						100.0%								100.0%	0.0%
RTMP							98.4%						1.6%	98.4%	<0.1%
SSH								100.0%						100.0%	0.0%
STUN									100.0%					100.0%	<0.1%
Shcast		1.0%			1.6%					97.5%				97.5%	0.0%
Skype											100.0%			100.0%	0.0%
Torrent									<0.1%			99.6%	0.3%	99.6%	<0.1%
WWW	<0.1%						<0.1%					<0.1%	99.9%	99.9%	**0.3%**
													Average:	97.7%	<0.1%

Table IV. Results of experimental validation (10-fold cross-validation). The tables present confusion matrices and performance metrics for the experiments 2-5 described in Section 5.2: a) domain name classification, b) port number classification, c) full DNS-Class, and d) performance after 8 months since training.

Prepared using nemauth.cls
DOI: 10.1002/nem