Computer Networks 56 (2012) 598–614

Contents lists available at SciVerse ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Extracting user web browsing patterns from non-content network traces: The online advertising case study ☆

Gabriel Maciá-Fernández [a,*], Yong Wang [b], Rafael A. Rodríguez-Gómez [a], Aleksandar Kuzmanovic [c]

[a] University of Granada, Dept. Signal Theory, Telematics and Communications, CITIC, Spain
[b] University of Electronic Science and Technology of China, Chengdu, China
[c] Northwestern University, Evanston, Illinois, USA

ARTICLE INFO

Article history:
Received 9 February 2011
Received in revised form 12 May 2011
Accepted 21 October 2011
Available online 29 October 2011

Keywords:
Online advertising
Web navigation
Web fingerprinting

ABSTRACT

Online advertising is a rapidly growing industry currently dominated by the search engine 'giant' Google. In an attempt to tap into this huge market, Internet Service Providers (ISPs) started deploying deep packet inspection techniques to track and collect user browsing behavior. However, these providers have the fear that such techniques violate wiretap laws that explicitly prevent intercepting the contents of communication without gaining consent from consumers. In this paper, we explore how it is possible for ISPs to extract user browsing patterns *without* inspecting contents of communication.

Our contributions are threefold. First, we develop a methodology and implement a system that is capable of extracting web browsing features from stored non-content based network traces, which could be legally shared. When such browsing features are correlated with information collected by independently crawling the Web, it becomes possible to recover the actual web pages accessed by clients. Second, we evaluate our system on the Internet and check that it can successfully recover user browsing patterns with high accuracy.

1. Introduction

Online advertising is a $20 billion industry that is growing rapidly [1]. Examples of online advertising include contextual ads on search engine results pages, banner ads, rich media ads, social network advertising, online classified advertising, advertising networks, and e-mail marketing. Google [2], who originally controlled 35% of the ad server market, finally acquired DoubleClick [3], a 34% market share holder, giving the combined online ad firm more than 69% of the market [4].

Internet Service Providers (ISPs) have for years looked on jealousy as Google has grown rich on their subscribers' web browsing, while the ISPs have been reduced to "dumb pipes," ferrying internet traffic for subscribers but unable to win their online spending [1]. In an attempt to reverse this trend, some ISPs started cooperating with companies such as Phorm [5], NebuAd [6], and FrontPorch [7]. These companies use deep packet inspection techniques, *i.e.*, inspect packets payload to intercept web page requests and responses generated by ISPs' subscribers as they roam the net. Then, they extract information related to users web navigation, and sell it to advertisers, allowing them to apply so-called behavioral ad targeting, *i.e.*, the ads sent to users are wisely chosen depending on the previous navigation patterns and preferences of these users.

☆ A preliminary version of this work appeared in Proceedings of IEEE Infocom 2010.

* Corresponding author. Address: Dept. of Signal Theory, Telematics and Communications. E.T.S. Computer and Telecommunications Engineering, University of Granada, c/ Daniel Saucedo Aranda, s/n, 18071 Granada, Spain. Tel.: +34 958241000 (20048); fax: +34 958 24 08 31.

E-mail addresses: gmacia@ugr.es (G. Maciá-Fernández), ywang@uestc.edu.cn (Y. Wang), rodgom@correo.ugr.es (R.A. Rodríguez-Gómez), akuzma@cs.northwestern.edu (A. Kuzmanovic).

1389-1286/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2011.10.012

A major problem in the above arrangement between ISPs and companies that deploy deep packet inspection based data collection systems is a *legal* one[1]: unlike Google (not a broadband provider), ISPs that provide broadband services are *not* exempt from "the Federal Wiretap Act, originally enacted in 1968 to protect against phone wiretapping and amended in 1986 to cover computer network communications. It states a simple prohibition: *thou shalt not intercept the contents of communications (see 18 U.S.C Section 2411(1))* [9]. *Violations can result in civil and criminal penalties.*" (extracted from [8]). Indeed, this prohibition has clearly been violated by deep packet inspection techniques [10]. The law predicts several exceptions, *e.g.*, security reasons (see 18 U.S.C. Section 2511(2)(a)(i) [9]) or user consent (see 18 U.S.C. Section 2511(2)(d) [9]) but behavioral advertising is certainly nowhere on the exception list.

Pressed by the legal constraints on one side, and by huge market opportunities on the other, ISPs (*e.g.*, [11,12]) started addressing the legal issue by altering their customer-service agreements to permit monitoring by describing it as "performance advertising services" [1]. Each company allows users to opt out of the ad targeting, though that permission is buried in customer service documents' footnotes. Other providers, such as AT& T and Verizon have pledged to refrain from tracking customer Web behavior unless they receive explicit "opt in" permission to do so [13]. Still, there is a strong concern that these approaches opt the user out of targeted ads, but *not* the online data collection. Hence, there is a fear that ISP-enabled ad targeting with deep packet inspection techniques is highly vulnerable to lawsuits [14], which is why many ISPs are reluctant to deploy this technology.

The main idea in this paper lies in abandoning controversial deep packet inspection techniques and try to reverse engineering user browsing patterns using alternative methods. We refer to the Electronic Communications Privacy Act [15–17] which defines the sharing of particular types of stored records of online activities. It states that any provider can hand-over *non-content* records to anyone except the government (see 18. U.S.C. Section 2702(c)(6) [16]). Consequently, sharing non-content-based stored headers – such as TCP headers – with anyone except a government body is legal [8].

The key challenge and the main research question we attempt to address then become if it is possible, and how accurately and scalably, to recover user browsing access patterns based solely on fairly limited information provided in TCP headers? We show that web browsing patterns stay highly visible at the TCP layer, and we design a method to automatically extract such features. Next, we profile the websites, building a fingerprint of every hosted web page by extracting relevant features such as object size, cacheability, location, link information, transfer modes, *etc.*. Finally, we design an algorithm that correlates the two sources of data to detect the pages accessed by clients.

We have evaluated our algorithm and show that it achieves high detection rates, *i.e.*, 86%, with false positive rates below 5%. The fundamental reason for such performance is its ability to extract and exploit significant statistical page diversity available at all sites we explored. Furthermore, we demonstrate that the algorithm is resilient to data staleness, *i.e.*, when either network traces or web profiles are outdated. While the page properties necessarily change over time, we show that a subset of unique properties remain, making the detection resilient with time.

We further show that the approach is resistant to different browsing scenarios including pipelining, caching, NAT-level flow multiplexing, and various browser versions. We also demonstrate that the algorithm scales to entire websites while preserving high detection performance. Finally, we evaluate our approach in the 'wild' and successfully recover browsing patterns based on real traces collected from a group of 17 volunteers.

This paper is structured as follows. We summarize the general framework of our approach in Section 2. We then extensively describe the details of our approach in Section 3. In particular, we provide the key ideas, the methodology and solutions. In Section 4 we present the results on the evaluation of the proposed approach in a controlled environment, while Section 5 presents the evaluation in a non-controlled environment. Next, we provide related work in Section 6. Finally, we discuss the results in Section 7 and conclude in Section 8.

2. The online advertising framework

In this section we provide an insight of the general framework in which our approach is applied, *i.e.*, a behavioral ad targeting business model where an ISP is willing to participate.

An example behavioral ad targeting scenario is as follows. Consider the Los Angeles Lakers web site. Based on its content, it posts ads for products a Lakers fan might be interested in: NBA store, Adidas, a Los Angeles hotel, *etc.* With information about users' navigation patterns, advertisers might consider that a person surfing at Lakers site is also a potential car customer because he recently accessed an article about car prices at the New York Times web site. This means that car companies might be interested in displaying an ad to that user at the Lakers site, too.

Here, three roles are distinguished. First, *advertisers* are those companies interested in advertising their products online. Second, *publishers*, which own web sites (in our example Los Angeles Lakers web site), participate in the business by publishing ads in their web pages. Finally, *commissioners* are intermediate companies that connect both advertisers and publishers and provide the technology to make it possible the ad distribution to the publishers' web sites.

A typical online advertising process is as follows. A client downloads a web page from a publishers' site. This page contains a pointer to a commissioner's server (typically a javascript code), which makes the client ask for an ad. Here, the commissioner is responsible for selecting an appropriate ad for that client and sending it. In the selection of an

[1] In this paper, we focus on the U.S. Federal Law. Still, many international laws are similar to the U.S. Federal law [8].

ad, a commissioner typically follows two possible strategies: (*i*) it takes into account information related to the contents of the web page solicited by the client (*contextual advertising*), and/or (*ii*) it considers the profile of the client (*behavioral targeting*), *e.g.*, navigation history, preferences, interests, etc.

In the context of behavioral targeting, certain companies offering end services, *e.g.*, Google, Amazon, are able to obtain useful information about client profiles through the information obtained from the own offered service. As an example, Google obtain navigation history of users by using click-through [18] when they search in the Google's search-engine; social networks are able to obtain client profiles when they share interests with their online social environment. For these "end-services companies", the availability of this information is practically straightforward and, therefore, they can potentially become a commissioner.

In this model, our approach would allow ISPs to participate in the online ad business model by obtaining such information from network traces and without having to use deep packet inspection techniques, in case they are reluctant to do this because of legal implications. Once the information about users navigation patterns is obtained, an ISP could either sell this "product" to current commissioners or it could even play the commissioner role. Now, we discuss how this information could be acquired by an ISP and the limitations of this information for being sold as *user navigation patterns*.

3. Recovering web browsing patterns from stored TCP headers

In this section, we introduce a methodology for recovering web browsing patterns from the information available in TCP headers. A web browsing pattern for a user is a list of web pages visited by that user. Here, we identify an user by his IP address, and a web page by its URL. Although the identification of user-IP address has some limitations, we discuss how it is still useful to have this information for the purposes of behavioral ad targeting.

An ISP which applies the methodology explained in what follows is expected to obtain a list of web pages (URIs) visited from different IPs. This information is the "product" to be sold or shared with commissioners in the online ad market.

Now, we present the necessary background on the topic. Next, we describe our approach and the corresponding algorithm. Finally, we discuss a method for handling several possible sources of error.

3.1. Background

Here, we present the necessary background on web browsing. Although the material presented here is common knowledge, we present it to define and clarify the terminology used in advance.

Web browsing is a process that relies on the client server paradigm. Using the Hypertext Transfer Protocol (HTTP) [19], a client (also termed as browser) sends an HTTP request to a web server soliciting a web page. Different resources in the web server are specified by different and unique Uniform Resource Identifiers (URIs) [20].

A web page typically consists of a *page file* and corresponding object files [21]. A page file is uniquely determined by a URI, this URI normally being that reached after a user either type it in the browser or follows a link in other page that points to it. We could say that the page file is really the container of the web page. A page file references other object files, called *objects*, that are inserted in the web page. For example, the page file identified by URI_1 in Fig. 1 references two objects, which could be images, media resources, or even scripts. Each object is also identified by an unique URI, *e.g.*, URI_{11} and URI_{12} in Fig. 1.

Independently from the HTTP protocol version, when accessing a web page, the corresponding page file is always downloaded first. Once a page file is downloaded, the browser parses its contents (typically HTML code or scripts) and extracts a list of URIs corresponding to the *objects* that should be downloaded to render the given page. Depending on the protocol version (*e.g.*, HTTP 1.0 vs. 1.1), these objects could be downloaded using one or more TCP connections, as we explain in detail below.

Objects referenced from a web page could reside at the same server as the page file. However, for reasons such as content distribution (*e.g.*, [22]), they could be hosted at other servers as well. If an object resides at the same server (determined by a unique IP address in our approach) as the page file, we term the object as *internal*. Otherwise, it is *third-party* [23].

In addition to referencing objects, a page file typically hosts *links* (pointers) to other web pages as well. They enable clients to access other web pages "by clicking" them at a given web page. Like objects, links could be *third-party* and *internal* depending on the location of the corresponding page file's URIs.

In our approach, given that both page files and objects are really referenced by URIs, we establish the following criterion to decide if a URI is a page file or not: *a given URI corresponds to a page file if and only if there exists a link on the given website pointing to that URI.* Note that a URI could be pointed by a link from a third-party website, not being linked from the internal web pages. In this case, we simplify our approach considering only those URIs pointed within the website. Furthermore, note that a web page does not always consist of multiple objects, and it does not necessarily point to other web pages. Indeed, a

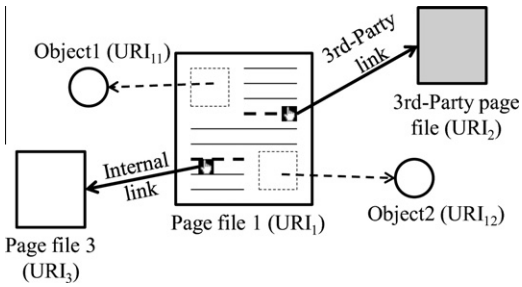

Fig. 1. Model for a web page structure. Pages are composed of a page file and objects. Page files contain references to objects and links to other pages.

web page can be a simple HTML file or an image. Then, it may occur that a page file (identified by URI_R) concurrently references and points (links) to a given URI, say URI_x. As a result, we consider that URI_x is an object from URI_R perspective, but also corresponds to a page file, because there exists a link on this website that points to URI_x.

3.2. Methodology

The problem we aim to solve is the following: "Given a packet-header network trace (without payloads), recover the web pages visited by users." The key idea is as follows: (*i*) Extract the Web-level communication features from the network-level information available in TCP headers; (*ii*) Profile Internet websites visited by users represented in the trace, *i.e.*, independently crawl the given websites and collect statistics about web pages, *e.g.*, object size, cacheability, locality, links among pages, *etc.*; and finally, (*iii*) correlate the information from the two sources to detect web pages actually accessed by clients.

3.2.1. Website profiling

To accurately and comprehensively profile a website, we extend the `wget` tool to build a web crawler. Contrary to the common web crawler application, *i.e.*, building a comprehensive and up-to-date replica of the content available on the Web for the sake of mining information from it (*e.g.*, [2]), our goal is different. We aim to extract characteristics about the web pages at a website that will later be used to recover actual user access patterns at that site. In particular, our crawler is capable of extracting the following information about each of the discovered URIs at a website.

- *Size.* A page file or an object corresponding to a given URI could be downloaded in either *plain* or *compressed* modes, depending on the browser and server settings. Our goal is to obtain the corresponding file size (in bytes) in both modes. Hence, the crawler makes two separate download attempts for each URI. First, with the `Accept-Encoding='gzip.deflate'` field set in the HTTP request, indicating a compressed mode; and second, without this feature, indicating the plain format. If the server does not support compressed downloads, the two sizes will be identical. It is important to understand that the estimated size accounts for the sum of the given object size *and* the corresponding HTTP header.
- *Cacheability.* Caching is an important web-related mechanism. It allows retrieving web page objects from intermediate repositories such as proxies, shared caches, or browsers. The browser and the server have mechanisms to decide if a given object should be cached or not [19]. Hence, objects in a web page could be *cacheable* or *non-cacheable*.
  In our approach, it is important to distinguish if an object is cacheable or not, mainly due to the fact that, if a cacheable object is inserted in a web page, the download of that page does not always imply the download of the object.
  The HTTP response header obtained from the server for a given URI allows the crawler to estimate if a page file or an object is cacheable or not. In particular, whenever the field `Cache-Control:no-cache` or the field `Pragma:no-cache` is present in the response header, this implies that the file specified by the given URI is non cacheable. Likewise, if none of the fields `Cache-Control:max-age`, `Expires`, and `Last-Modified` appear in the header, the given page file or object is considered as non cacheable [19].
- *Locality.* The crawler records the location of each page file and its corresponding objects. The location could be *internal*, *i.e.*, the page file or the object is hosted at the same server (same IP address) as the website. Otherwise, the location is *third-party*.
- *References.* A page file pointed to by a URI could contain references to other URIs, corresponding to objects referenced by this page file. The crawler parses this page file and extracts a list containing these references.
- *Links.* A page file pointed to by a URI could contain links to other URIs. The crawler parses this page file and extracts a list containing these links. It further crawls URIs corresponding to internal links, *i.e.*, page files.

3.2.2. Extracting web browsing features from network traces

Here, our goal is to extract the Web-level browsing features from network traces; in particular, the number of web pages accessed by a client and the size and location of page files and objects corresponding to these web pages. When combined with the information obtained via web profiling (Section 3.2.1), these features will enable recovering user web browsing patterns (Section 3.2.3). We refrain from mining packets payloads to obtain URIs accessed by clients or content generated by servers since both approaches violate the Federal Wiretap Act [8]. Indeed, we constrain ourselves to recording and later inspecting TCP headers only.

One-way TCP header collection. Our approach is tailored towards access ISPs, and it requires an ISP to record TCP packet headers at a tapping point in the network. While it is generally possible to obtain data in both (client–server and server-client) directions in access networks, that is typically not the case in non-access networks due to path asymmetry [24]. Still, our approach is applicable even in such scenarios because it requires collecting TCP headers in the *single* direction only, *i.e.*, from clients to servers, as we explain in detail below. To extract HTTP level communication from the trace, we filter out traffic on port 80 and create per source IP subtraces.

Web page-based trace slicing. Our next goal is to further separate each of the user subtraces into separate *slices*; each slice should ideally consist of packets that correspond to a single page accessed by a client. To achieve this goal, we exploit the well-known web-user behavior. In particular, it has been shown experimentally that either a machine [25] or a Web user [26] requires at least one second to process and react to the display of a new page. (We experimentally verify this result ourselves in Section 5 by evaluating a representative user browsing data set we collected.) As a result, each user 'click' at a link on a website is followed by a period of activity corresponding to a web page download, and a period of inactivity corresponding to the page processing. Hence, we use these moments of user inactivity to separate the user traces into

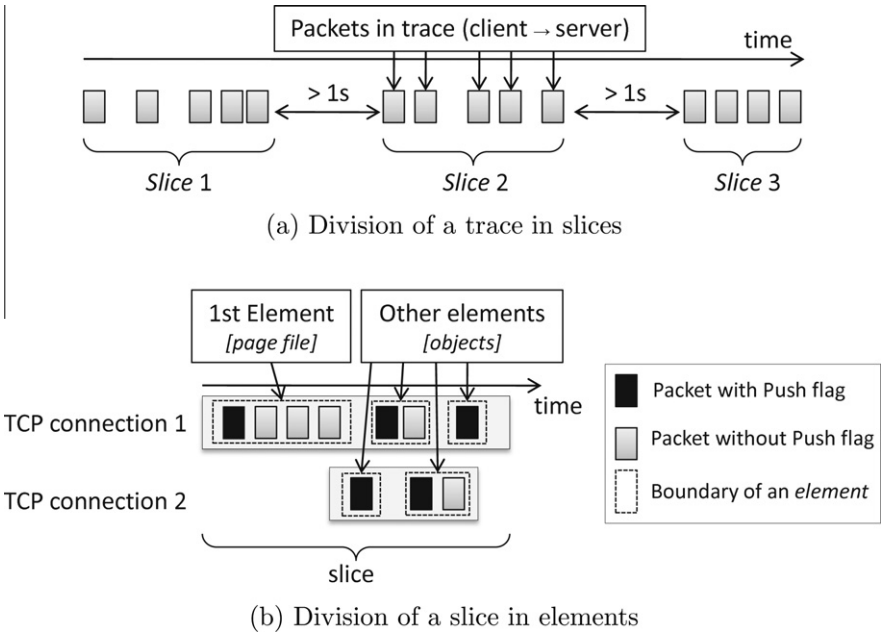

Fig. 2. Model of the trace. Time intervals higher than 1s divide the trace in *slices*. The *elements* contained in a slice are separated by packets with TCP PUSH flag set within a TCP connection.

slices. Even when this is not the case that more than one page can end up in a slice, our algorithm can handle this case as well, as we demonstrate below. Thus, in our model of the trace, slices are composed of several *elements*, each one containing the request for the download of either a page file or an object for a certain web page (see Fig. 2(a)).

Extracting web page features. Our next goal is to extract the number of elements (page file and objects) within a given slice, and their size and position (internal vs. third-party). This information will later be used, among other features, to detect the actual web pages accessed by clients in the trace (Section 3.2.3). To recover the number of elements and their size and position, we inspect the TCP packet headers.

The overall model of a trace slice is shown in Fig. 2(b). Here, three issues are considered for our objective. First, as indicated above, when accessing a web page, the corresponding page file is always requested and downloaded first. This means that the first element detected in the trace would correspond to the page file. The rest of elements are considered objects referenced by that page file. Second, each HTTP request for any of the web page objects is requested in a *separate* TCP packet (except when pipelining is enabled. See Section 3.3.2 for this case). Third, in the vast majority of scenarios, TCP packets carrying HTTP requests have the TCP PUSH flag set.[2]

The TCP PUSH flag [27] exists to ensure that the data is given the priority. In particular, when either a sending or receiving TCP receives a TCP packet with TCP PUSH flag set, they must immediately act: send a TCP packet to the other end in the sender case, or pass the data to the receiving process in the receiver case [27]. This particular flag is used quite frequently at the beginning and end of a data transfer, affecting the way that the data is handled at both ends. In the HTTP context, Web browsers set TCP PUSH flags when sending HTTP requests, both for page files and objects.

Next, to estimate the size of elements within a trace slice, we proceed as follows. We consider that the TCP packets corresponding to an element are those belonging to the same TCP connection and are delimited by two consecutive TCP PUSH enabled packets. Once the page file and different objects contained in every slice have been identified, we extract their sizes from the sequence numbers available in TCP ACK packets. Finally, we determine the object location, *i.e.*, internal vs. third-party, in a straightforward way.

Several issues, including the ability to estimate object boundaries (*e.g.*, due to pipelining) and the file size estimation accuracy (*e.g.*, due to variable HTTP header size), exist. We analyze these issues in depth in later parts of the paper (Sections 3.3 and 4.5). Below, we present an algorithm capable of using information extracted from TCP-level network traces to recover the actual web pages accessed by clients.

3.2.3. Detection algorithm

Here, we present an algorithm that correlates information obtained via website profiling (Section 3.2.1) and features independently extracted from TCP headers (Section 3.2.2) with the goal of detecting actual web pages accessed by clients in the trace. The algorithm is independently executed on each *slice* of the trace. To avoid confusion, we will refer in advance to page files and objects for the data

[2] Even when TCP PUSH flag is not set in the TCP header, HTTP requests could be distinguished based on the TCP packet size, which is greater than the TCP ACK size.

obtained from the website profiling phase, while the objects downloaded in the trace will be termed as *elements*.

Denote by $E = \{E_1, E_2, \ldots, E_l\}$ the set of l elements identified in a trace slice. Next, denote by $P = \{P_1, P_2, \ldots, P_n\}$ the set of n web pages identified at the given website in the web profiling phase. Further, denote by $PF = \{PF_1, PF_2, \ldots, PF_n\}$ the set of n page files associated with the identified web pages. Also, denote by $O_i = \{O_{i1}, O_{i2}, \ldots, O_{im}\}$ the set of m different objects contained in page P_i. As we explained above, each page file PF_i or object O_{ij} can be downloaded in either plain or compressed mode. Then, while the size (in bytes) of an element is denoted by $S(E_i)$, for a given PF_i or O_{ij}, generically X, we denote by $S_1(X)$ its size in the plain mode, and by $S_2(X)$ its size in the compressed mode. While the sizes $S_1(X)$ and $S_2(X)$ are obtained in the website profiling phase, $S(E_i)$ is calculated during the trace features extraction phase.

The algorithm is executed in the following three phases, also summarized in Fig. 3 and detailed in Table 1:

Filtering phase. A subset of the pages from the set P could be eliminated in a straightforward manner. Indeed, if the number of *non-cacheable* objects at a given page P_i is greater than the number of elements in the slice, l, then the page P_i is eliminated from the set.

Tagging phase. During this phase, for each element E_K from the set of identified elements E, we compare the size $S(E_k)$ and the location $L(E_k)$ (internal/third-party) of the elements in the trace slice separately with the size and location of all page files and then objects in the website profile. This allows us to identify possible candidate web pages to be selected as downloaded. Each page file or object whose size and location corresponds to that of one of the elements is tagged as *identified* (Table 1, tagging phase, steps 1(a) and 1(c)). Moreover, if E_k is identified with a single page file or a single object in their respective comparisons, it is also tagged as *unique* (Table 1, tagging phase, steps 1(b) and 1(d)). Because a unique object/page file is present in only one page of the website, its identification makes this page a good candidate to have been downloaded. Finally, in this phase, all the pages with identified page files are compiled in a set P_{PF} and those with identified objects in a set P_O.

Selection phase. The selection phase takes the set of pages P_{PF} and P_O as input data and aims to decide which pages are downloaded in the trace, and hence should be included in P_D (initially empty, Table 1, selection phase, step 1). We distinguish two different cases: (*i*) if unique page files and objects have been identified in the slice, all the pages that contain them are selected (Table 1, selection phase, step 2). Indeed, because multiple web pages might be present in the slice (Section 3.2.2), selecting pages with unique characteristics leads to high detection rates in such scenarios, as we will demonstrate in Section 4.5.3 below. (*ii*) However, in case that no unique page files or objects are identified, we make a best effort to minimize false positives (pages detected by the algorithm which have not been really downloaded); hence, in this case our goal is to identify a *single* page in the slice.

We apply the following strategy. First, we consider only those pages, if any, that are present in both P_{PF} and P_O; that is, $P_{PF} \cap P_O$. Indeed, if there is an overlap between the two sets, it is likely that a page from the overlap has been

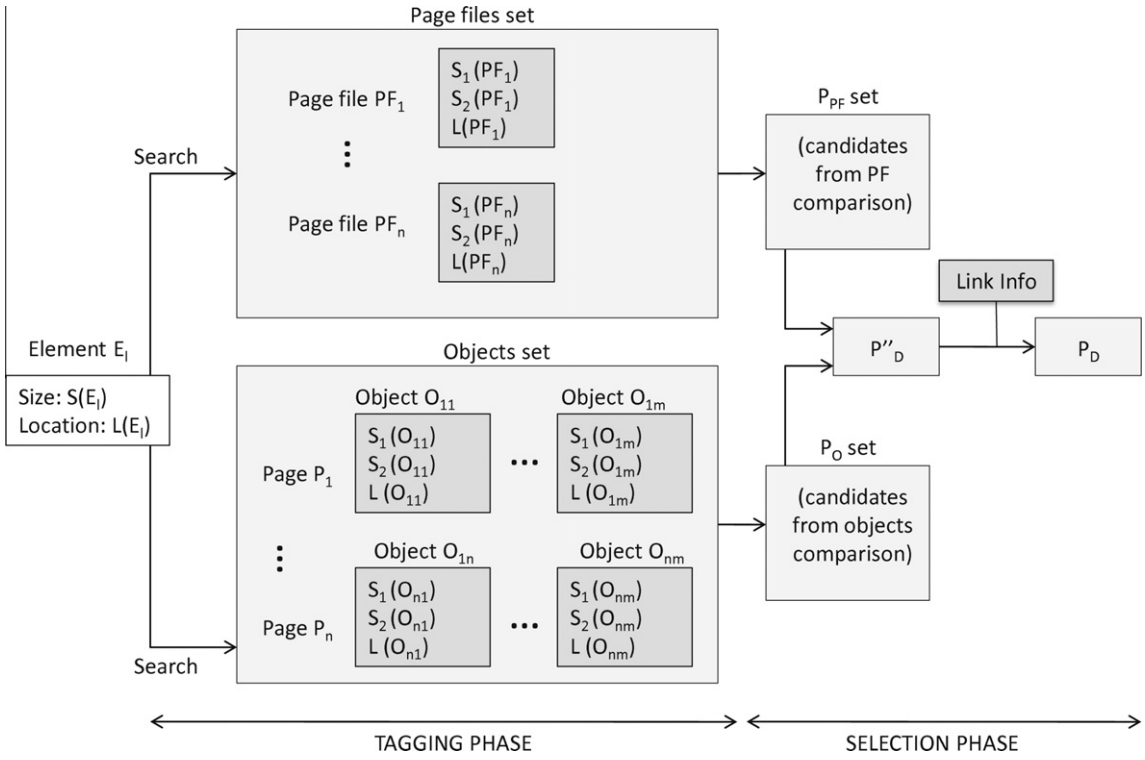

Fig. 3. Overview of the steps in the detection algorithm for a single element E_l. Filtering phase has been omitted for its simplicity.

Table 1
Summary of the steps in the detection algorithm.

Filtering phase
1. Eliminate all pages from set P for which the number of *non-cacheable* objects is greater than the number of elements in the slice

Tagging phase
1. For all $E_k \in E$:
   (a) For all $PF_i \in PF$, check if $S(E_k) = S_m(PF_i)$ and $L(E_k) = L(PF_i)$:
      i If true $\rightarrow$ PF_i= identified
   (b) If only one PF_i= identified$\rightarrow$PF$_i$ = unique
   (c) For all O_{ij}, check if $S(E_k) = S_m(O_{ij})$ and $L(E_k) = L(O_{ij})$:
      i If true $\rightarrow$ O_{ij}= identified
   (d) If only one O_{ij}= identified $\rightarrow$ O_{ij} = unique
2. Build sets P_{PF} and P_O with pages with identified page files/ objects respectively.

Selection phase
1. Initial set of detected pages: $P_D = \emptyset$
2. If pages in $P_{PF} \cup P_O$ contain any unique page files/objects, P_D= pages in $P_{PF} \cup P_O$ with any unique page files/objects. End of algorithm
3. $P'_D = P_{PF} \cap P_O$. If $P'_D = \emptyset \rightarrow P'_D = P_{PF} \cup P_O$.
   (a) If $S(P'_D) = 1, P_D = P'_D$. End of algorithm
4. Obtain P''_D selecting from P'_Dthose pages with highest percentage of identified objects
   (a) If $S(P''_D) = 1, P_D = P''_D$. End of algorithm
5. Obtain P'''_D filtering P''_D with link information
   (a) If $S(P'''_D) = 1, P_D = P'''_D$. End of algorithm

P_O: Pages with identified objects, P_{PF}: pages with identified page files, E: set of all elements in the trace, E_k: element k in trace, PF: set of all page files in the website, PF_i: page file i, O_{ij}: object j in page i, $S_m(X)$: size type m of page file/object X, $S(\cdot)$: size operator, $L(\cdot)$: location (internal/third-party) operator, $\emptyset$:empty set.

accessed. However, if there is no overlap, we are unable to reduce the set, and hence we consider all the pages in both sets $P_{PF} \cup P_O$. The resulting set is P'_D (Table 1, selection phase, step 3). If more than a single page still remain, we filter P'_D and extract only page (s) with the highest percentage of identified objects in the considered slice, *i.e.*, set P''_D (Table 1, selection phase, step 4).

If several candidates still remain, we consider the user navigation pattern. In particular, we use the simple heuristic that if a user accesses more than a single page at a website, it is likely that there exist links from one page (hence one slice) to the next page (hence next slice) accessed by the client. Thus, among the remaining candidates in ith slice, we only choose those that are linked from the pages in the P_D set obtained for the previous $(i-1)$th slice. The resulting set is P'''_D, (Table 1, selection phase, step 5). During steps 3–5, if any set P'_D, P''_D or P'''_D contains only one page, it is selected as the final decision P_D (Table 1, selection phase, steps 3(a), 4(a), and 5(a)). Otherwise, if several candidates still remain, they are all discarded in order to minimize the false positives.

3.3. Dealing with sources of errors

Errors are inevitable part of a detection process. In particular, the algorithm can incorrectly identify as accessed web pages that have not really been downloaded by clients in the trace (false positives), or fail to detect the pages actually accessed by clients (false negatives). Below, we emphasize the key factors responsible for false detection. First, we summarize the key elements that lead to inaccuracies in a web object size estimation. Then, we outline other factors that can impact detection accuracy. We evaluate all these factors and their impact on the detection accuracy in Section 4.

3.3.1. Object size estimation

The estimate of an object (or a page file) size obtained (*i*) via website profiling (Section 3.2.1) and (*ii*) via TCP-level headers (Section 3.2.2) can be different. Whenever such a difference occurs, the probability that the algorithm will make a false decision increases. The key factor contributing to the difference in the estimated object size is the potential variability in the HTTP header size. We provide several examples below.

First, an HTTP request may include a cookie. Although the size of a cookie is usually constant, in some cases its length depends on a seed used for its generation, which might involve parameters such as nonces, timestamps, or source IPs. In order to reduce the amount of false positives due to cookies, our crawler considers two different sizes for those pages that return a cookie: one taking into account the cookie size and another without it. Moreover, when cookies are used, they are typically inserted in a page file, which is sufficient to identify a user. Because the transfer of subsequent objects is most likely cookie-free, the impact of cookies is limited.

Second, an object might be downloaded using the chunking transfer mode [19]. Indeed, when a server does not know in advance the final size of the content that it is sending, the sender breaks the message body into chunks of arbitrary length, and each chunk is sent with its length prepended [19]. Hence, the complete size of the object depends on the number of chunks used and their own size. As a result, it can happen that subsequent requests to the same non-cacheable objects on the same site can generate different HTTP header sizes.

Third, the HTTP header size (and hence the entire web object size estimate) depends on both the server and the client setup. As an example, the use of persistent TCP connections [19] depends not only on the server, but on the browser configuration as well. As a result, given fields in the HTTP header may appear or not, *e.g.*, '`Connection: keep-alive`' field. Finally, even if a given header field exists in two HTTP responses, the values present in the header fields might be different. For example, '`Timeout: 99`' and '`Timeout: lOO`' headers differ in one byte. We explore the effects of all the above factors on false positive and negative rates in Section 4.

3.3.2. Other sources of error

Here, we outline other factors that can lead to detection inaccuracies. We evaluate each of these issues separately in the next section.

Dynamic website behavior. Websites can change over time. For example, a site administrator can change the content of a given page. The relevant question thus becomes: How frequently do page files or objects at a website change, and how does that affect the ability of the algorithm to detect such pages? We explore this issue in depth in Section 4.4 below.

Pipelining. HTTP1.1 proposes pipelining, *i.e.*, send subsequent HTTP requests within a single TCP connection without waiting for the corresponding HTTP responses. This approach blurs the visibility of object boundaries at the TCP level and complicates the corresponding web object size estimation. While pipelining is not widely spread in the Internet, as we explore later in the paper, a relevant question is how our algorithm performs when pipelining is enabled. We explore this issue in Section 4.5.1.

Caching. All objects belonging to a page are not always downloaded from the server. While we explicitly address this issue in the algorithm, the question is how does this mechanism affect the accuracy of the algorithm. We explore this issue in Section 4.5.2.

Overlapping page downloads. Several factors can generate so-called overlapping page downloads to appear in a single trace slice. First, inter-click estimation might not always be fully accurate. Hence, it can happen that two or more web page downloads from the same website can end up in the same trace slice. Second, when Network Address Translation (NAT) boxes are used, a number of clients behind the NAT will have the same source IP address visible at the tapping point. While accurate per-client trace slicing is still feasible using destination (server) IP addresses, it is possible that at given time intervals, one or more clients behind the NAT concurrently access the same website. Third, a single user can (nearly) concurrently access several pages at a single website. All these issues lead to the overlapping page downloads effect. We explore our algorithm's performance in such scenarios in Section 4.5.3.

Spurious requests. During the navigation process, certain spurious HTTP requests that do not correspond with a page download can be generated. These are mainly caused by client web-based applications, *e.g.*, google toolbar, live search toolbar, or by AJAX scripts embedded in web pages. In some cases, these requests can be filtered by considering the usual destination IP addresses, *e.g.*, google server. AJAX scripts, on the other hand, are a well-known challenge even for the latest commercial crawlers. Hence, these requests will interfere with the detection process generating false positives.

4. Evaluation

Here, we evaluate our approach in a number of challenging, yet realistic scenarios. In particular, we explore the resilience of our algorithm when either a web profile or a network trace is outdated. Then, we explore the issues of pipelining, caching, overlapping page downloads, and the browser diversity.

4.1. Experimental setup

Before presenting the performance evaluation, we first explain how we obtained two necessary datasets – crawled website logs and TCP-level traces. To emulate a realistic setup, in which the two datasets are typically obtained from two different points in the network, (*i.e.*, TCP-level traces collected from an ISP network, and crawled logs by a different set of machines), we proceed as follows. We collect the two datasets from two different parts of the world, one in Europe and the other one in USA. In all scenarios, we use a crawling spider we designed to profile the websites; we generate network traces using the `Firefox 3.0.5` browser, with default parameters, *i.e.*, caching enabled and pipelining disabled. We explore other browsers and parameter settings in Section 4.5.4 below.

Website profiling. We select six different websites shown in Table 2. The sites range from newspapers, sports, commercial, and educational sites. While this is certainly a small fraction of the Web, our key goal is to understand in-depth performance of our algorithm in diverse scenarios. In the next section, we perform experiments in the 'wild' and evaluate our algorithm by crawling a larger number of websites. Still, the selected sets represent a fairly diverse set of websites in terms of their dynamic properties, *i.e.*, how frequently do they change, and static characteristics, *i.e.*, what are the page files and objects features with respect to locality, uniqueness, *etc.*.

In each of the sites, we crawl a subset of pages, *i.e.*, 2,000 web pages (except for Univ2 which has less than 2,000 pages). We select this threshold because it enables us to crawl all six websites within a 24 hours interval. This helps us to understand how our algorithm performs when either a web profile or a network trace is outdated, an issue we explore in depth in Section 4.4 below. Finally, in order to understand how the size of a website impacts the results, we crawl one of the websites in full in Section 4.6.

TCP-level traces. To obtain TCP-level traces (using the Wireshark tool [28]), we emulate user behavior by creating quasi-random walks over 100 out of the 2,000 pages at a website; we call these 100 pages *test pages*. In particular, we at random select a page out of the 2,000 pages; then, we randomly select the next page from the set of pages that the given page links to. When no links exist from a given page, we randomly select another page from the set, and continue the quasi-random walk until we collect 100 pages. Finally, we compute detection statistics as we explain in Section 4.3 below. For all experiments, we collect ten independent test sets, and show averages. We move beyond emulation in Section 5 and deal with real user browsing traces.

Table 2
Experimental websites.

Websites
NYtimes (http://www.nytimes.com)
FC Barcelona (http://www.fcbarcelona.com)
IKEA (http://www.ikea.com)
Toyota (http://www.toyota.com)
Univ1 (http://www.northwestern.edu)
Univ2 (http://ceres.ugr.es)

4.2. Site uniqueness

Here, we show the statistics for unique-size page file and object in the six websites. Such files are invaluable in the detection process since their presence in a trace uniquely identify a web page.

Fig. 4 shows the percentage of pages with unique objects, unique page files, and with either unique objects or unique page files. The figure shows that the percentage of pages with unique objects is high, except for the two universities. This is because commercial or news websites are usually rich with pictures and other objects, which dramatically increase the page diversity. For example, in the IKEA website, many pages have a unique picture showing different products.

The figure shows that the percentage of pages with unique page files is high in all web sites. Indeed, even when the web pages share the same template, they still have different text resulting in different page file sizes. Moreover, the percentage of pages that either have unique size objects or unique page files is necessarily even higher. These high percentages indicate that the use of unique-size objects or page files is a powerful feature.

We use these statistics to explain the basic performance of our algorithm. In particular, from the statistical point of view, the percent of pages with unique objects could be considered as the (loose) lower bound of the expected success rate, and the percentage of pages with unique size objects or unique page files is considered as the (loose) upper bound.

4.3. Basic performance

Here, we explore the performance of our algorithm for the six websites. We apply the methodology explained in Section 4.1 above, *i.e.*, use 100-page long test sets to compute the success rate, false positives and negatives. In the figures here and in the rest of the paper we show the success rate and false positives. (False negatives could be computed as 100% – success rate (%)).

Fig. 5 shows the results. We make several observations. First, the success rate is around 86% on average over the websites, and false positives are below 5%. In all cases, the success rate is above the lower expected bound, as we predicted above. Moreover, in certain scenarios (IKEA, Toyota, Univ1, and Univ2), the performance is even above the upper expected bound. This is because we made expectations only based on the site uniqueness. Still, other issues, such as the use of link information, can further improve the results even in scenarios when no unique items are detected at a website.

The performance for Barcelona and NYtimes is approximately between upper and lower bounds. In both cases the reason for not reaching the upper-bound performance is due to effects explained in Section 3.3.1. In particular, we experienced increased chunking-mode transfers in the NYtimes case for page files. Nevertheless, other factors, such as unique objects, the percent of identified objects and link relationships, keep the performance above the lower expected bound. More concretely, in the NYtimes case, success rate of 84% (Fig. 5) surpasses the lower expected bound of 71% (Fig. 4).

4.4. The role of time scales

Both network traces and web profiles could be outdated for a number of reasons. For example, several days might pass until an ISP ships its traces to an advertising company. Likewise, crawling the Web is an exhaustive process. Hence, several days or more can pass until a crawler revisits a site and updates its profile. Here, we evaluate how these issues impact the accuracy of our algorithm.

Methodology. We select 100 pages as the *preliminary test set* for each website in the first day of the experiment. Then, we crawl the given sites once a day for one week, and collect a new 2,000 pages profile each day for each of the sites. Because some of the websites change over time, the 2,000 pages that we obtain are not always the same. As a result, the initial test set also reduces in some cases. Although the pages crawled on the first day typically

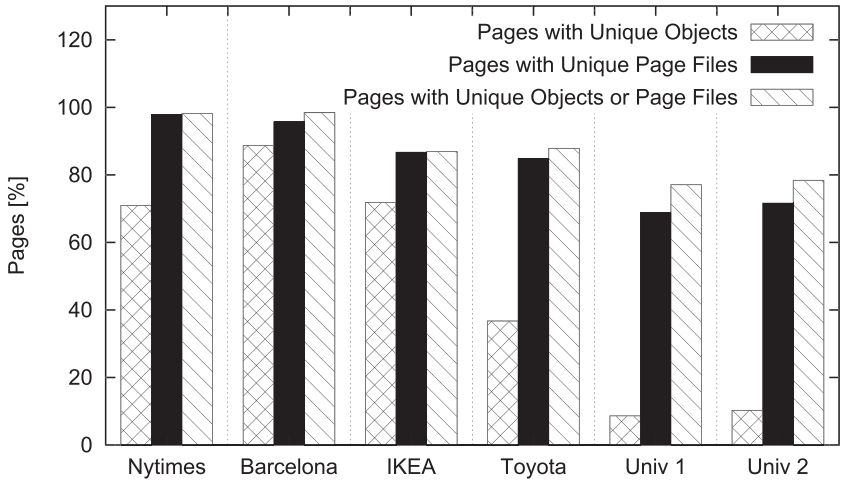

Fig. 4. Evaluation of the uniqueness of page files and objects in the experimental websites.

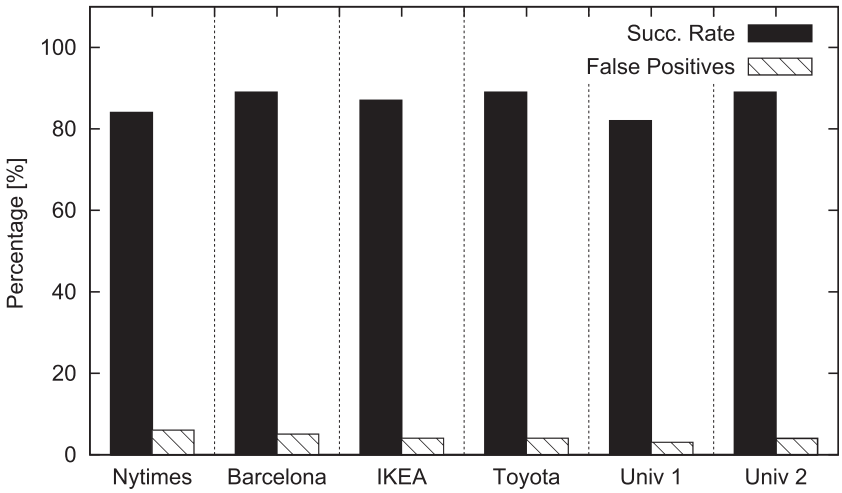

Fig. 5. Basic performance of the algorithm for the experimental websites.

still exist on the website, our limited crawling process does not manage to download these pages. Hence, we proceed in two steps. First, we determine the pages that exist during the entire period and consider them as a *final test set*. Second, we explore how the pages in the final test set change over time.

Fig. 6 illustrates that, during the seven day period, the number of overlapping pages stays the same in Toyota and universities, suddenly dives a bit in NYtimes and Barcelona, and gradually decreases in IKEA. Again, all the web pages from the first day are typically available on the web site, the overlap decrease is due to the limited number of crawled pages and the addition of new pages. More specifically, Toyota's updates are relatively the slowest as its products are usually coming out over longer time scales. On the other side, NYtimes may add many pages in its website in one day, which leads to a huge shrink in the overlapping size. IKEA, as an in-between case, slowly updates its website and hence the number of overlapping pages decreases at the same pace. As a result, for NYtimes, Barcelona, and IKEA cases, the size of the final test set is 81, 76, and 98 pages respectively, while for Toyota and universities cases, the size is 100 pages.

Finally, we divide the six websites in two categories. The first one includes sites that have the final test set less than 100 pages (NYtimes, Barcelona, and IKEA). For this set, we capture the TCP-level trace at the last day of the experiment (day 7 in Fig. 6). The second set includes sites that have the final test set equal to 100 pages (Toyota and universities). For this set, we capture the TCP-level trace at the first day of the experiment (day 1 in Fig. 6). In the former scenarios, the website profiles are out of date. In the latter scenarios, the TCP-level traces are stale. In the experiment, we compare the TCP-level traces with web profiles taken during the seven day period.

Performance. Figs. 7(a) and (b) show the success and the false positive rates (computed over the final test) as a function of time. The reference point in each figure (day 0) corresponds to the time when TCP-level traces are obtained. As a result, day 0 in Fig. 7(a) corresponds to day 1 in Fig. 6. Likewise, day 0 in Fig. 7(b) corresponds to day 7 in Fig. 6.

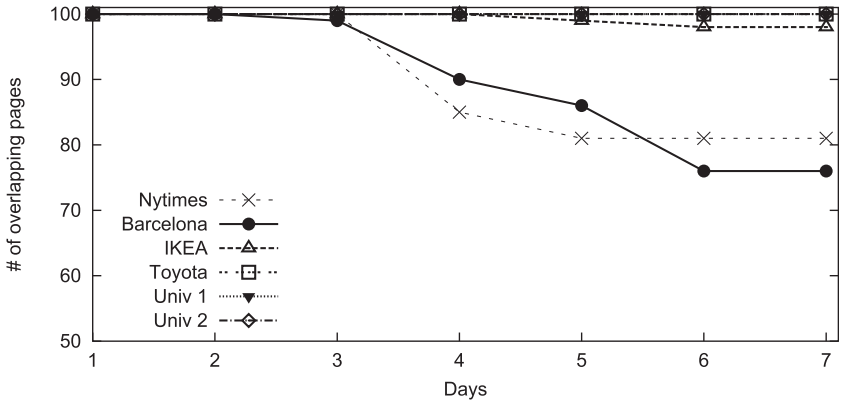

Fig. 6. Evolution of the number of overlapping pages for the experimental websites.

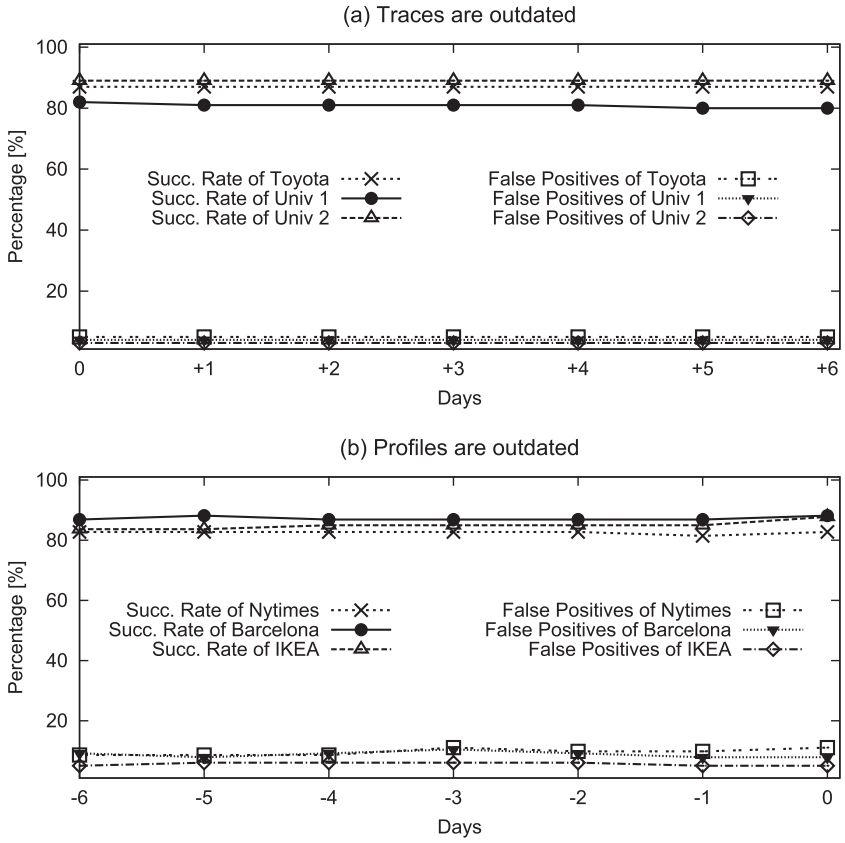

Fig. 7. Success and false positives rates obtained by the algorithm as a function of time.

Fig. 7 provides three insights. First, in Toyota and the universities (Fig. 7(a)), the success rate stays almost constant; in other scenarios (Fig. 7(b)), the success rate changes marginally. For example, the success rate of Barcelona drops from 88% to 87%. Second, in all cases, the success rates reach the peak on the day when the test TCP-level trace is collected because the properties of page files and objects in the crawled profiles are more likely to be the same as those in the TCP-level trace on the same day. Third, besides the success rates, false positives are also resilient with time. In Fig. 7(a) the change rate of false positives remains same; In Fig. 7(b) the false positives change smoothly; for example, in the IKEA case, the minimum is 5% and the maximum is 6%.

Change rates. To understand the causes of the above observations, we explore the change rates of page files and objects size. In the experiment, we compare the page files and objects size of pages in the final test set with web profiles taken during the seven day period. We define the change rates as the percentage of inconsistency of page files or objects size between the final test set and web profiles. We also consider the objects that are permanently removed from the given pages as changed, *i.e.*, their size becomes zero.

Fig. 8 shows the page file and object change rate as a function of time. The first finding is that the change rate of both page files and objects is much smaller in Fig. 8(a) than in Fig. 8(b). This is caused by the same reasons discussed with respect to Fig. 6 above. For example, in the Toyota case, the web administrators update their web news if some new products are available in the market, which typically happens over longer time scales. On the contrary, the websites are updated much more frequently for news and other commercial websites such as NYtimes, Barcelona, and IKEA.

Second, the change rate increment is the largest within one day from when the traces are taken, *i.e.*, day +1 in Fig. 8(a) and day −1 in Fig. 8(b). After that, there is almost no change, *i.e.*, for days 2–6 in Fig. 8(a) and days −2–−6 in Fig. 8(b). This is because a part of pages, like main pages, updating the top-line news or the latest product promotions at commercial websites is typically updated frequently, not all the pages.

Finally, in all cases, the change rate of page files is higher than the objects size change rate. For example, In NYtimes case, the page files change rate of 60% highlights the above fact that news web pages, particularly the text part, are frequently updated. At the same time, the change rate for objects is less than 3% at day −6. Thus, despite a highly dynamic site behavior, our algorithm is capable of accurately detecting the given web pages with high accuracy, as we demonstrated in Fig. 7. This is because a subset of web pages' unique properties remain consistent over time.

4.5. Different browsing scenarios

Here, we explore different browsing scenarios. Thus, we evaluate how (*i*) pipelining, (*ii*) caching, (*iii*) overlapping

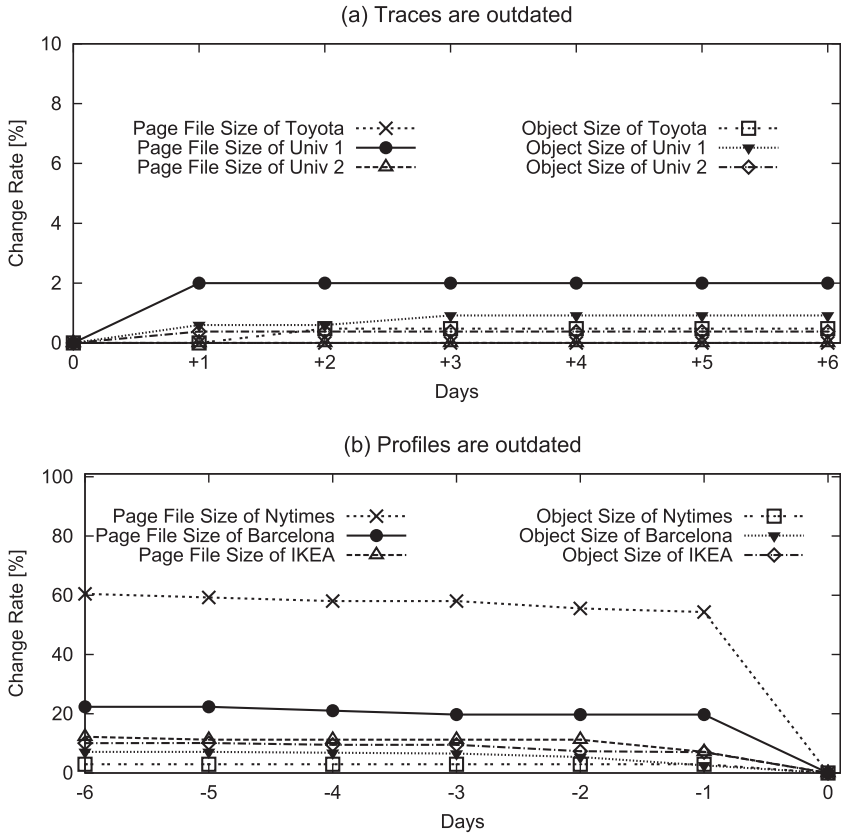

Fig. 8. Change rate of the web pages in the six studied websites.

page downloads, and (*iv*) different browsers affect the performance. The first three experiments thus far are conducted using the `Firefox 3.0.5` browser with its default settings, *i.e.*, caching enabled and pipelining disabled by default. We conduct all experiments on the Toyota server, using the above methodology. To avoid dynamic effects explored above, we collect all traces on the same day.

4.5.1. Pipelining

We first explore how widely pipelining is spread in the Internet by analyzing a Tier-2 network trace with 153,583 HTTP requests. We identify the existence of several HTTP requests in the same TCP segment as a pipelining signature. Our results show that the percentage of pipelined segments is smaller than 1%, while the percent of users that use browsers with pipelining enabled is around 2% of the total number of users (in terms of source IP addresses). We find these figures reasonable, as the use of pipelining depends on browser configuration and capabilities. In fact, some of the most widely used browsers have not activated the pipelining feature by default, *e.g.* `Firefox`; others do not support pipelining, *e.g.* `IE7` and `Google Chrome`. To the best of our knowledge, `Opera` is the only browser that enables pipelining by default.

Despite low usage of pipelining, clients might be tempted to enable this feature in order to prevent ISP-based ad targeting. We explore whether such an attempt would be successful. We test our approach when pipelining is used in a `Firefox 3.0.5` browser.

Table 3 shows the results. We can see that there is only a slight difference in the results, as the success rate degrades by 1% only. The reasons are the following. First, the fact that a browser enables pipelining does not imply that all HTTP requests will be pipelined (for performance reasons), but only a subset of them. Indeed, only 12% of the TCP segments containing HTTP requests are really pipelined by the browser. As a result, the bulks of the objects sizes are correctly identified. Second, even if larger percents of objects would be pipelined, the requests for page files cannot be pipelined. This is because a browser does not know in advance which objects to fetch before

Table 3
Performance evaluation for different browsing scenarios.

Scenario	Success rates (%)	False positives (%)
Pipelining disabled	89	4
Pipelining enabled	88	4
Cache disabled	90	4
Cache enabled	89	4
Sequential	89	4
Parallel-two	74	7
Parallel-four	63	8

it downloads the page file. Hence, high detection rates are still feasible.

4.5.2. Caching

In this experiment we evaluate the effects of browser caching mechanisms. We consider two scenarios: (*i*) navigation without caching, in which we disable the cache in the browser and (*ii*) navigation with caching, in which we enable the cache in the browser.

Table 3 shows the results for the two scenarios. As expected, we can see that results when cache is disabled are better, *i.e.*, the success rate increases to 90%, while false positives remain unchanged relative to the caching scenario. The slight improvement in the performance is due to fact that the existing non-cacheable elements (typically all page files and a subset of objects) already create a strong inter-page diversity. Nevertheless, more information in the non-caching scenario produces a better result.

4.5.3. Overlapping page downloads

Overlapping page downloads means that more than one web page might end up in a single trace slice. For example, this can happen either due to NAT-induced effects or inaccurate inter-click time estimation. While we show in the next section that none of the two effects are likely to happen, we nevertheless explore our algorithm's performance in this case. For this, we have emulated the download of a test set of pages with three different navigation patterns: (*i*) pages have been downloaded without overlapping (*sequential browsing*), (*ii*) two different pages are downloaded simultaneously (*parallel-two browsing*), and (*iii*) four different pages are downloaded simultaneously (*parallel-four browsing*).

Table 3 shows the results. As expected, the performance is the best in the sequential case, when there is only a single page in a slice. While the success rate necessarily degrades when the number of pages increases per slice, it is still quite reasonable (74% in parallel-two and 63% in parallel-four). These results are mainly due to the step 2 of the selection phase (Table 1), which takes advantage of unique page files and objects from multiple pages.

4.5.4. Different browsers

We experiment with different browsers. In particular, we obtain different traces using `Firefox 3.0.5`, `Internet Explorer 7.0`, and `Google Chrome`. All the browsers disable pipelining and enable caching. We have not found any differences in the performance of the algorithm when using the three traces obtained. This implies that our approach is independent from different browser types.

4.6. Scaling the website profile

To evaluate how our approach behaves with increased website profile, we crawl the entire Toyota site and download 9,211 pages. Then, we repeat the experiment by repeating the procedure explained above.

Our results show that the success rate is resilient with the increase of the website profile. More specifically, the success rate of Toyota reduces from 89% to 81%. At the same time, the false positives increase from 4% to 8%. We investigate this result in more depth, and find that 78% of pages have either unique size objects or unique page files, while this percentage was about 88% when the website profile was 2,000 pages long (Fig. 4). Additionally, each page in Toyota site has 97.3 links on average which reduces the ability of our algorithm to sweep out many incorrect results.

5. Performance in the wild

Here, we further evaluate our approach by using real user browsing patterns at websites of their own choice. Our experiment is done in two phases: (*i*) Collecting URI-level traces from a group of volunteers during a long period of time. We evaluate the real behavior of traffic coming out from users. (*ii*) Capturing a real trace of requests directed to a real production server. We assess the behavior of our approach when all the traffic for a server is considered.

For the first experiment, we collect URI-level traces from 17 volunteers (with their consent) from USA, Europe and Asia during 1 month. These traces contain users' anonymized identifications, as well as the visited URIs and their corresponding timestamps. From this information, we select 40 different websites with the highest number of requests. The given websites cover a wide range of interests, from online news to computer hardware, game hardware and software, cell phones, sports equipment, jewelers, movies, and online publications. For each website we build its profile. Then, we choose the list of URIs as the test set in our experiment. A TCP level trace is obtained by replaying the user navigation patterns (visited URIs and timestamps) within these sites.[3]

As a result we obtain a success rate of 85% and a false positive ratio of 9%, slightly higher than the result obtained in the controlled environment. This demonstrates that our approach works well in the wild with a reduced group of people and a medium size number of websites.

With our second experiment in the wild we explore the detection algorithm capabilities in a scenario where we collect access traffic to a website. We capture the web requests of the server labeled as 'Univ2' in Table 2 during 13 days. We present the results obtained for this experiment in Fig. 9. Here, the day zero represents the day in which the crawling of the website is done. This crawling is used to obtain results from our algorithm during the six previous days, *i.e.*, days $[-6,-1]$ in Fig. 9(a). Likewise, we obtain the results for the six subsequent days, i.e., days [1,6] in Fig. 9(b). We can see that the results are quite similar to those shown in Fig. 7. While the false positive ratio is between 8% and 10%, as in the results from Fig. 7, now the success rate is inside the range 80–84%, slightly lower than the obtained in the controlled environment.

In summary, we conclude that results obtained in a real environment are very similar to those presented in the controlled experiments. There are slight reductions of accuracy mainly due to certain effects like the presence

[3] We compress all inter-access times longer than one minute to one minute.

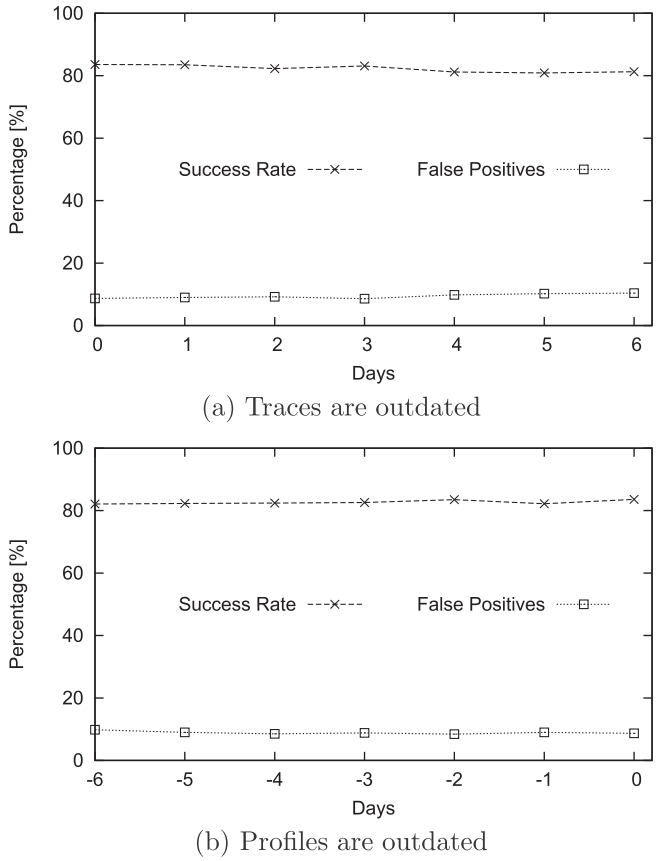

Fig. 9. Success and false positives rates of the algorithm when applied in a real environment to Univ2 server.

of fast navigation of a certain percentage of users. In particular, we have detected in our real world traces a percentage of requests (3.8%) that have been sent to the server, but the user blocked its download and visited another page before the download of the previous one was completed.

NAT behavior. To understand NAT-like behavior, we further explore URI-level traces from ten people from the same local network. We find that users' interests are diverse since they access 9,183 domains in total, while only 565 of them (6.15%) are accessed by more than one user. Moreover, only 61 domains (0.66%) are accessed by five users. We then study the requests sent to the most popular website (a total of 16,756 requests) to discover the number of simultaneous accesses based on their timestamp. These simultaneous petitions generate what we have called overlapping pages downloads (Section 4.5.3). Considering that this happens when more than one user accesses the most popular site within the same second, we find that only 0.44% of the accesses are simultaneous. In summary, the presence of NAT boxes will not degrade the performance of our detection method since its impact is small.

Inter-click time. Finally, we verify the inter-click time statistics in order to validate our choice of 1 s for slicing the trace (Section 3.2.2). We process 315,444 timestamps in total and 94.53% of them have the inter-click time larger than 1 s.

6. Related work

Encrypted Web traffic. Our work relates to the security research efforts aimed towards analyzing and inferring encrypted web browsing traffic [29–33]. The authors of these papers have demonstrated that it is feasible to reveal the sources of encrypted web traffic despite encryption. In light of this finding, they further analyze additional mechanisms that can help secure such communication. The key differences between our work and this thread of papers are threefold. (*i*) We have shown that there are incentives to reveal user browsing patterns even when they are not encrypted. As a result, the scope of the problem changes from the one covering a small fraction of encrypted web pages on the Web [29–33] to the *entire* Web 'landscape'. This dramatic change of scope in turn fundamentally impacts both (*ii*) our methodology and (*iii*) the range of potential counter mechanisms, as we elaborate below.

Regarding methodology, our approach differs from the security-oriented related work in three aspects. First, because we operate in the 'wild', unlike previous work, we consider multiple web features characteristic for 'open'

web communication. This includes object location, uniqueness, cacheability, link information, different transfer modes, distinction between page files and objects, *etc.*, to characterize web pages. Second, we add mechanisms that consider possible sources of error that are inevitably created by the state-of-the art web practices (see Section 3.3). Finally, contrary to previous work (*e.g.*, [33]) that has severe scalability issues, our approach can effectively scale. Indeed, we have demonstrated that it is feasible for an ISP to successfully collect TCP headers (and recover user browsing behavior) anywhere in the network, even behind a proxy or a NAT (Section 5). In addition, because the destination IP address is known in the advertising case, we effectively reduce the scalability problem from the entire web space to a *single web server*. Moreover, because we are capable of statistically characterizing a website in a more comprehensive way, we effectively scale the detection process.

Regarding counter-eavesdropping mechanisms, the approaches proposed in the related security work (*e.g.*, padding) might not work entirely in the advertising case. This is primarily due to rich site characteristics (*e.g.*, the use of CDNs), which still leak page-identifiable information from the sites. More importantly, websites have no incentives to apply any countermeasures in the advertising scenario. Indeed, they are one of the primary beneficiaries that make money from online advertising [1]. Hence, we argue that in the case of advertising, a comprehensive legislative reform is the only realistic way to address this problem.

Traffic analysis. Karagiannis et al. [34] propose a method to recover application types without considering any payload and ports information. Felten and Schneider [35] demonstrate that web servers can use the inter-arrival time of HTTP requests for objects on a web page to reveal the presence of items in the browser's cache. Others also use timing information and packets sizes to either identify sources of information in obfuscated flows [31], discover sensitive security information [36], or classify application services without looking at payloads [37]. In our work we go one step beyond, and attempt to detect the exact web pages accessed in a TCP flow, also without accessing the payload, even when it is not encrypted.

Coull et al. [38] suggest a technique for obtaining destination addresses of flows in anonymized traces and even sensitive information about the routing structure of the network. This work is similar to ours in the sense that they get advantage of sources of information other than their own trace. In their case, they use information extracted from DNS servers and search engines, while our approach crawls websites to get additional information.

7. Discussion

Too much crawling? In order to obtain web profiles, our approach requires large-scale web crawling. Hence, the question is whether this overhead can hinder the potential deployment of this approach. There are three issues here. First, we demonstrate that even when web profiles are not fully up to date or when they do not fully cover all pages in a website, the approach is still valid. Second, not all sites are equally interesting from the advertising perspective. As an example, .edu domain might be less interesting than. com domain. Hence, less crawling than in the search engine case is needed. Third, even within a site, crawling could be restricted to only the most relevant pages. This could be done, for example, by selecting only those pages which have a significant pagerank value [39].

Nevertheless, significant crawling is certainly needed. Advertising is a $20 billion industry, and any ISP that attempts to enter this market (in a legal way) should be ready to invest sufficient resources.

Storing URIs from HTTP headers? There is ambiguity on whether a URI should be treated as a part of content or not [8]. There are no ambiguities, however, that TCP headers are not the part of the content though [8] and hence could be legally shared. Still, we have demonstrated that it is possible to recover the content of communication (specified by a URI) without directly observing it. This example again clearly shows not only that a comprehensive legislative reform is needed, but that the Internet's development creates novel legal challenges on almost daily basis.

/24 annonymization is insufficient. Sharing TCP header-level traces by removing the lower 8 bits from both source and destination addresses is typically considered sufficient to preserve user privacy. At the same time, this approach enables meaningful networking research. Our initial experiments indicate that such traces could still be used for ad targeting towards the source/24 address space. The key issue is that the number of web pages within a/24 network address block still enables *per-page* identifiability with reasonable false positive rates (below 10%) in most cases.

Dynamic IP addresses. Dynamic IP addresses are an inherent problem for ISP-enabled advertising (including deep packet inspection), because it blurs the per-source IP identifiability. There are two issues. First, an access network can still keep the track of individual users since the access ISP assigns such addresses. Non-access networks are still able to recover user browsing properties, yet ad targeting is possible only at aggregate basis.

8. Conclusions and future work

In this paper, we showed how it is possible to recover user web browsing patterns *without* inspecting the packet payload. By extracting HTTP-level 'reflections' available at the transport layer, and by profiling web sites in a comprehensive way using page files, objects, different transfer modes, linking information, cacheability, and locality, we designed an algorithm capable of effectively merging the two data sources and discovering web pages accessed by clients. We evaluated our methodology on the Internet using both emulation and real user browsing patterns.

Our key insights are the following: (*i*) The development of the Web in recent years, *e.g.*, rich image mixtures and the use of CDNs, has created a significant statistical diversity among web pages at a website, making them highly identifiable. (*ii*) The page identifiability remains high even

when a trace from an ISP is outdated, or when the web profile is not fresh due to crawling limitations. Even though the page features can dramatically change over time, we showed that a sufficient subset of identifiable features does stay available. (*iii*) The detection process is resilient to a number of challenges, including pipelining, caching, NAT-level multiplexing, different browser types, and it effectively scales. (*iv*) Endpoint-based countermeasures are highly limited; not only because it is hard to comprehensively cover rich inter-page diversity, but because websites have no incentives to apply such countermeasures since they are one of the primary beneficiaries of the advertising business.

As shown, the performance of the detection process presented in this paper depends on a good and extensive crawling process (see discussion in Section 7). Thus, the development of an efficient crawler adapted for the needs of our methodology would constitute an interesting future work contribution.

An additional future work is related to the need for the development of a theoretical framework which allows to relate the functioning of the proposed approach with the model of a web site profile. This work should model the different sources of noise for this approach, *i.e.*, caching, compression, cookies, embedded dynamic code, variable headers sizes, etc. As a result, it would potentially allow the estimation of the expected detection rate and false positives/negatives rates.

Acknowledgments

This work is supported by Spanish MEC project TEC2008-06663-C03-02 (70% FEDER funds), NSF CAREER Award No. 0746360, and China Scholarship Council.

References

[1] Washingtonpost.com: Every click you make, <http://www.washingtonpost.com/wp-dyn/content/article/2008/04/03/AR2008040304052.html>.
[2] Google, <http://www.google.com/>.
[3] Double click, <http://www.doubleclick.com/>.
[4] Internet marketing news: Doubleclick deal means Google controls 69% of the online ad market, <http://www.browsermedia.co.uk/2008/04/01/doubleclick-deal-means-google-controls-69/-of-the-online-ad-market/>.
[5] Phorm, <http://www.phorm.com/>.
[6] NebuAd, <http://www.nebuad.com/>.
[7] Frontporch, <http://www.frontporch.com/>.
[8] P. Ohm, D. Sicker, D. Grunwald, Legal issues surrounding monitoring during network research (invited paper), in: ACM IMC '07.
[9] 18 united states code 2511, <http://www4.law.cornell.edu/uscode/html/uscode18/usc_sec_18_00002511—-000-.html>.
[10] ISP behavioral targeting v. you, <http://www.seoserpent.com/2008-09/isp-behavioral-targeting/>.
[11] Embarq, <http://www.embarq.com/>.
[12] Wide open west, <http://www1.wowway.com/>.
[13] Washingtonpost.com: AT& T, Verizon to refrain from tracking users online, <http://www.washingtonpost.com/wp-dyn/content/article/2008/09/25/AR2008092504135.html?hpid=sec-tech>.
[14] Behavioral advertising could be illegal: NebuAd leaves ISPs vulnerable to wiretap, privacy laws, <http://www.dslreports.com/shownews/94578>.
[15] "18 united states code 2701, <http://www.law.cornell.edu/uscode/html/uscode18/usc_sec_18_00002701—-000-.html>.
[16] 18 united states code 2702, <http://www.law.cornell.edu/uscode/html/uscode18/usc_sec_18_00002702—-000-.html>.
[17] "18 united states code 2703, <http://www.law.cornell.edu/uscode/html/uscode18/usc_sec_18_00002703—-000-.html>.
[18] G.-R. Xue, H.-J. Zeng, Z. Chen, Y. Yu, W.-Y. Ma, W. Xi, W. Fan, Optimizing web search using web click-through data, in: ACM Proceedings of the CIKM '04, 2004, pp. 118–126.
[19] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, Hypertext Transfer Protocol – HTTP/1.1, Jun. 1999, Internet RFC 2616.
[20] T. Berners-Lee, R. Fielding, L. Masinter, Uniform Resource Identifier (URI): Generic syntax, Jan. 2005, Internet RFC 3986.
[21] F. Donelson Smith, Félix Hernández Campos, Kevin Jeffay, David Ott. "What TCP/IP protocol headers can tell us about the web." SIGMETRICS Perform. Eval., Rev. 29, 1, pp. 245–256, June 2001.
[22] Akamai, <http://www.akamai.com/>.
[23] B. Krishnamurthy and C. Wills. Privacy diffusion on the web: a longitudinal perspective, in: Proceedings of the 18th International Conference on World wide web (WWW '09). ACM, New York, NY, USA, 2009, pp. 541–550.
[24] H. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, A. Venkataramani, iPlane: An Information Plane for Distributed Services," in OSDI '06.
[25] M. Crovella, A. Bestavros, Self-similarity in world wide web traffic: evidence and possible causes, IEEE/ACM Trans. Networking 5 (6) (1997).
[26] B.A. Mah, An empirical model of HTTP network traffic, in: INFOCOM '97.
[27] Transmission control protocol, RFC 793.
[28] Wireshark, <http://www.wireshark.org/>.
[29] H. Cheng, R. Avnur, Traffic analysis of SSL encrypted web browsing, 1998.
[30] A. Hintz, Fingerprinting websites using traffic analysis, in: Workshop on Privacy Enhancing Technologies '02.
[31] M. Liberatore, B.N. Levine, Inferring the source of encrypted http connections, in: ACM CCS '06.
[32] S. Mistry, B. Raman, Quantifying traffic analysis of encrypted web-browsing, 1998.
[33] Q. Sun, D.R. Simon, Y.-M. Wang, W. Russell, V.N. Padmanabhan, L. Qiu, Statistical identification of encrypted web browsing traffic, in: IEEE Computer Society SP '02.
[34] T. Karagiannis, K. Papagiannaki, M. Faloutsos, BLINC: Multilevel traffic classification in the dark, in: ACM SIGCOMM '05.
[35] E.W. Felten M.A. Schneider, Timing attacks on web privacy, in: ACM CCS '00.
[36] D.X. Song, D. Wagner, X. Tian, Timing analysis of keystrokes and timing attacks on SSH, in: USENIX SSYM '01.
[37] C.V. Wright, F. Monrose, G.M. Masson, On inferring application protocol behaviors in encrypted network traffic, J. Mach. Learn. Res. 7 (2006) 2745–2769.
[38] S.E. Coull, C.V. Wright, F. Monrose, M.P. Collins, M.K. Reiter, Playing devils advocate: inferring sensitive information from anonymized network traces, in: NDSS '07.
[39] <http://www.mypagerank.org/>.

Gabriel Maciá–Fernández is an Associate Professor in the Department of Signal Theory,Telematics and Communications of the University of Granada (Spain). He received a MS in Telecommunications Engineering from the University of Seville, Spain, and the Ph.D. in Telecommunications Engineering from the University of Granada. In the period 1999–2005 he worked as a specialist consultant at 'Vodafone España'. His research interests are focused on computer and network security, with special focus on intrusion detection, reliable protocol design, network information leakage and denial of service.

Yong Wang is a Ph.D. student in School of Computer Science and Engineering at University of Electronic Science and Technology of China. He received his BS degree from Shengda College, Zhengzhou University in 2003 and M.Sc. degree from University of Surrey in 2004 respectively. His research topics lie in the area of computer networking with emphasis on behavioral targeting in Web advertising field, location-based Web advertising, and the measurement of Web advertising infrastructure.

Rafael A. Rodríguez-Gómez is a Ph.D. student in the Department of Signal Theory, Telematics and Communications of the University of Granada (Spain). He received his M.Sc. degree in Telecommunications from the University of Granada in 2008. His research interests are focused on network security and more specifically on defense against DoS attacks, security in P2P networks and defenses against botnets.

Aleksandar Kuzmanovic is an Associate Professor in the Department of Electrical Engineering and Computer Science at Northwestern University. He received his B.S. and M.S. degrees from the University of Belgrade, Serbia, in 1996 and 1999 respectively. He received the Ph.D. degree from Rice University in 2004. His research interests are in the area of computer networking with emphasis on design, measurements, analysis, denial-of-service resiliency, and prototype implementation of protocols and algorithms for the Internet. He received the National Science Foundation CAREER Award in 2008.