Journal of Network and Computer Applications 99 (2017) 17–27

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

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

TBDClust: Time-based density clustering to enable free browsing of sites in pay-per-use mobile Internet providers

Luis Miguel Torres[a], Eduardo Magaña[b,c,*], Daniel Morató[b,c], Santiago Garcia-Jimenez[b], Mikel Izal[b,c]

[a] Naudit High Performance Computing and Networking S.L., calle Faraday 7, 28049 Madrid, Spain
[b] Public University of Navarre, Departamento de Automática y Computación, Campus Arrosadia, 31006 Pamplona, Spain
[c] Institute of Smart Cities, calle Tajonar s/n, 31006 Pamplona, Spain

ARTICLE INFO

Keywords:
Clustering TCP connections
Time-based density clustering
DBSCAN
Mobile web browsing
Online monitoring
Real traffic dataset

ABSTRACT

The World Wide Web has evolved rapidly, incorporating new content types and becoming more dynamic. The contents from a website can be distributed between several servers, and as a consequence, web traffic has become increasingly complex. From a network traffic perspective, it can be difficult to ascertain which websites are being visited by a user, let alone which part of the user's traffic each website is responsible for. In this paper we present a method for identifying the TCP connections involved in the same full webpage download without the need of deep packet inspection. This identification is needed for example to enable free browsing of specific websites in a pay per use mobile Internet access. It could be not only for third party promoted websites but also portals to gubernamental or medical emergency websites. The proposal is based on a modification of the DBSCAN clustering algorithm to work online and over one-dimensional sorted data. In order to validate our results we use both real traffic and packet captures from a controlled environment. The proposal achieves excellent results in consistency (99%) and completeness (92%), meaning that its error margin identifying the webpage downloads is minimal.

1. Introduction

The web is probably the Internet application that has grown and evolved the most during the past two decades. The simple and mostly static webpages of the 1990s have given way to much more complex sites. This complexity is caused, in the first place, by the addition of a wide variety of content types (like videos, scripts or interactive media) to the text and images that classic webpages hosted. However, modern websites not only offer these new content types, but they do so in a very dynamic way, keeping their content updated and tailoring their offer to each specific visitor. Services like e-mail, video streaming, on-line games or e-learning are, in many cases, provided through the web, taking advantage of the fact that web browsers are present in almost any network-enabled device and that web traffic usually faces few network restrictions. This ever-increasing popularity of the web has introduced new network requirements which have pushed for improvements in the web application protocols and the development of new techniques, like content distribution networks (CDNs) (Fortino and Mastroianni, 2009). As a consequence, the web has achieved a remarkable flexibility which allows it to provide a huge range of different services, but adding many layers of complexity in order to achieve it.

All these changes have obviously affected the profile of web traffic. Recent studies (Charzinski, 2010; Fang et al., 2016; Weinreich et al., 2008; Ihm and Pai, 2011) show that its characteristics have greatly changed from the (simpler) ones described in the 1990s (Catledge and Pitkow, 1995) for HTTP/1.0. This is partially the result of the introduction of HTTP/1.1. Persistent connections and pipelining have made obsolete the notion that every connection comprises a single request/response pair. CDNs have also made obsolete the notion of downloading the whole web page from a single server. Accessing a webpage requires nowadays downloading resources from multiple servers, due to the distribution of root webpage files, advertising banners, scripts for user tracking or multimedia content files. From a network traffic perspective, the whole webpage download implies a variable number of TCP connections with different durations and sizes, to multiple server IP addresses that will compose what we call a full *webpage download*. As a sizable amount of the content is dynamic,

* Corresponding author at: Public University of Navarre, Departamento de Automática y Computación, Campus Arrosadia, 31006 Pamplona, Spain.
E-mail addresses: luismiguel.torres@naudit.es (L.M. Torres), eduardo.magana@unavarra.es (E. Magaña), daniel.morato@unavarra.es (D. Morató), santiago.garcia@unavarra.es (S. Garcia-Jimenez), mikel.izal@unavarra.es (M. Izal).

http://dx.doi.org/10.1016/j.jnca.2017.10.007
Received 16 February 2017; Received in revised form 21 June 2017; Accepted 3 October 2017
Available online 05 October 2017

these connections may change if the webpage is accessed at a different time, by a different user, from different location, etc. Moreover, the same servers can be shared between different webpage providers, as happens with traditional shared hosting or with CDNs. This means that not even the server IP addresses can be used to identify a particular webpage download.

Identifying all the TCP connections involved in a webpage download is a functionality needed for example in the case of a mobile ISP (Internet Service Provider) offering a scheme of pay-per-use and providing free browsing to strategic commercial websites (Free Facebook, 2017). Another use case could be the whitelisting of institutional or emergency websites where availability could be of crucial importance and should be at no cost.

If we consider the set of TCP connections opened by a web browser during a period of time, it is far from trivial to ascertain which webpage download has originated each connection. In this paper we address this problem by presenting a method capable of *identifying individual full webpage downloads by clustering related connections together in real time*. This identification of the connections belonging to the same full webpage download could be implemented in web browsing control systems, proxys, and firewalls, where different treatments are necessary when browsing through different websites.

Traditionally, the method to control access to different sites has been to provide a white or black list of domain names or IP addresses to which the users are allowed access or not. All the IP addresses from all the servers that host files for a certain service have to be added to the white list. Those lists are hard to maintain because of the complex distribution of contents along different servers and CDNs. A change in the linked elements from a webpage could result in a new server where they are located. In order to allow access to this server, its IP address must be added to the white list. This requires updating the lists with a growing and changing number of IP addresses over time. This is neither scalable nor maintainable.

The most straightforward methods to cluster related connections together are based on Deep Packet Inspection (DPI). Using DPI the content of the HTML files can be parsed and the links to external files collected. Ad-hoc access rules can be added to the networking elements in order to allow access to the external content based on these links. However, deep packet inspection has multiple drawbacks: (1) it requires processing a lot of information for each connection, hindering real-time operation; (2) not all of the required URLs are available in the HTML file as some of them are dynamically created by Javascript code; (3) accessing user data raises privacy concerns and, depending on the local legislation, it may be illegal (Macia-Fernandez et al., 2012); (4) a sizable fraction of web traffic nowadays is HTTPS, for which application-level data is encrypted (Finley, 2014); and (5) encrypted traffic is indeed expected to become more and more prevalent in the future with HTTP/2 using transport layer security (TLS) by default (Belshe et al., 2014).

These considerations have motivated us to work with connection data at the transport level. Our proposal uses no DPI and it requires only to specify the main domain name or IP address of the website. The clustering of related connections with the main one will provide which connections must be allowed in order to render the full webpage or which traffic must be billed or not. Only the first packet for each TCP connection establishment will be used in the clustering process. This will allow very fast classification of TCP connections, enabling operation in real time even for heavy loaded links. Equivalently, NetFlow-type records as described by the IPFIX protocol (Claise, 2008) can be used in cases where real time operation is not necessary. These NetFlow records are easy to collect, store and process by routers and switches. NetFlow offers a very summarized description of each transport connection, providing source and destination IP addresses and ports, total bytes and packets in each direction, start timestamp, and duration. These fields provide more than the necessary information offered by the first packet in the connection. As far as we know, the problem of clustering connections related to the same webpage download using such small information and in real time operation has not been tackled by the scientific community.

In order to test our identification methodology we have used both real traffic traces and automatic captures of webpage downloads. After performing a thorough characterization of these captures and testing different approaches to our problem, we present a method based on the DBSCAN (Ester et al., 1996) clustering algorithm which was designed for density-based clustering in noisy databases. It is suited for our purposes as most of the connections from a webpage download are usually very close in time and there is a lot of noise in the form of automatic web connections initiated by other programs running in background (e.g. operating system updates, online cloud storage, antivirus updates, etc.). The results show that the proposal is able to cluster together a very high percentage of TCP connections belonging to the same web download (92%). It also reduces the effect of background connections, considering them as noise and obtaining an almost perfect consistency of the created clusters.

The rest of this paper is structured as follows. Section 2 provides a brief compilation of related work. Section 3 presents the Time-Based Density Clustering (TBDClust) algorithm for webpage downloads. In Section 4 we select the best values for the parameters in the algorithm and validate its correct behavior using real traffic datasets. Section 5 evaluates the performance and real-time operation of the algorithm. Finally, Section 6 concludes the work.

2. Related work

Web traffic characterization has been a popular field of study over the years, and works in the literature have taken different approaches to the problem. From a server perspective, some authors have focused on modeling the behavior and habits of the users that access the web server (Liu and Kešelj, 2007). Other researches have taken a user-centric point of view with works focused on application-level operation (Khandelwal, 2013) or the characteristics of the downloaded content (Zink et al., 2009; Butkiewicz et al., 2014). For example in Xie et al. (2013), a model for reconstructing web-surfing activities from packet traces is proposed. Other works have characterized specific types of web applications, usually with the intention to classify their traffic (Schatzmann et al., 2010; Schneider, 2008). All those analysis rely on DPI, which has serious drawbacks in scenarios with high load or encrypted traffic. There exist proposals without DPI requirements for traffic classification, like for example the one presented in Lu et al. (2016), but they do require some insight into the traffic profile for each application (packet sizes, inter-arrival patters, etc), which they obtain by analyzing a large quantity of packets from each flow, which takes a toll in the throughput they can achieve.

Clustering and other machine learning techniques have been used in multiple works for traffic classification purposes (Nguyen and Armitage, 2008; Callado et al., 2010; Zhang et al., 2015). However, the intent on most of those proposals has been to identify the application that generated the traffic, rather than trying to identify elements inside the traffic of a single application. The latter is precisely our case: we focus on the web application and we intend to find groups of related connections in its traffic. For unencrypted connections, this can be done by inspecting HTTP data, checking from which document each object was linked. However, we are interested in methods that do not require DPI. Some proposals have tried to address this problem. Bianco et al. (2009) used clustering techniques to identify user-sessions in web traffic. They defined a session as a set of TCP connections generated by a user while browsing through multiple webpages during a period of time. They used a mix of a hierarchical clustering and partitioning approach in a three-step process applied to connection start times. It was usable only for offline analysis because of its design and computational complexity. Other proposals require access to server logs and they group all the request from a single user into a browsing session for pattern analysis (Fatima et al., 2015).

In this paper, we focus on real time clustering of connections related with a single webpage download, with limited information provided by the first packet of each TCP connection (connection start times). Macia-Fernandez et al. (2012) introduced a method that intended to find individual webpage downloads in user traffic. Their approach was directed at identifying specific webpages for advertising purposes. It required a previous characterization of said webpages and relied on DPI. By contrast, the method we present in this paper is able to find individual webpage downloads by using only basic information from the first packet of each TCP connection and without requiring any previous characterization of the webpages.

3. Methodology for a webpage clustering algorithm

We consider that a *webpage download* comprehends all the data exchanged between a web browser and one or multiple web servers in order to render a specific webpage. From a traffic perspective, a webpage download comprehends the connections opened by the web browser after a user types a URL address or follows a hyperlink to a new webpage. These connections carry the content of said webpage or are somehow triggered by accessing it (e.g. Google Safe Browsing, 2017). We focus on connection parameters that can be used to identify which connections belong to the same webpage download.

In this section we present an evaluation of the information available from the network traffic that could be used in a clustering procedure. After this evaluation we formulate the new clustering algorithm proposal

3.1. Webpage downloads from a traffic perspective

Through this work, the client IP address is used to identify connections belonging to the same webpage download. The server IP addresses are not as useful for our purposes as they may seem intuitively. A single server IP address can host information for different websites like in a CDN or third-party provider scenario (Charzinski, 2010; Butkiewicz et al., 2011). Modern sites download content from a big number of different servers, a substantial fraction of which belong to third-party services that may be shared with other websites. This suggests that modern websites are less and less centralized in easily identifiable servers.

Another parameter of interest is the connection start timestamp. The connections that take part in the download are usually opened very close in time in order to provide the best user experience. Therefore, start timestamps are very interesting for clustering purposes and they will be discussed later.

The connection end timestamp is another candidate parameter: two connections from the same webpage download have a high probability of ending close in time. However, in most cases this is an effect of the connections also starting close in time. Non-persistent connections are usually very short and persistent connections have lengths defined by persistence timeouts in the client (the same timeout for all connections from the same browser) or in the server (usually the same default timeout for connections to the same server software). It is probable that two connections from the same webpage download would start close in time and have a similar duration, thus ending also close in time. The only case in which end timestamps offer non-redundant information is when a user closes the web browser or the navigation tab. All connections related to the webpage that are still open will be closed at this time. However, it is difficult to know how many webpages (tabs) the user is closing at the same time.

Finally, the last connection parameter to consider, the connection size (in bytes or packets), is useless, as connections for the same webpage download are very variable in size. Although some services tend to produce larger downloads (e.g. video streaming), they will also use smaller ones for the accessory content.

The timestamps in all packets and connection sizes do not allow clustering on the fly. Therefore, the start timestamps are the most promising connection-level parameters for identifying connections that belong to the same webpage download. This parameter can be extracted from the first packet of the TCP connection which is easily identified by looking for packets with only the TCP SYN flag activated. We discuss this parameter intuitively below.

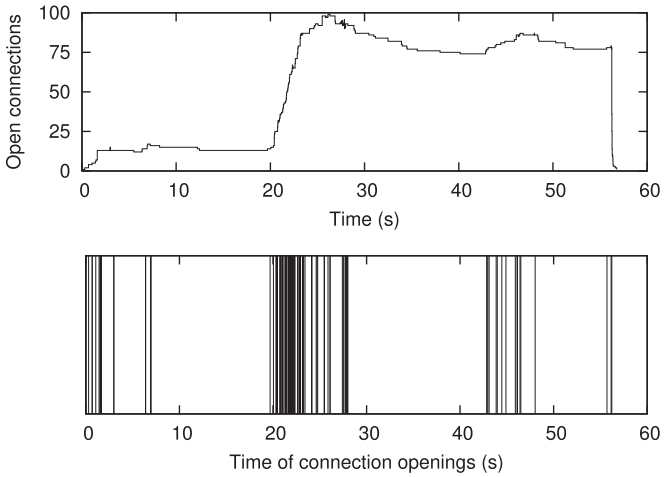

Fig. 1. Start timestamps for the example web session with 3 webpage downloads: 0–10 s facebook, 20–30 s newspaper, 40–50 s newspaper again.

Lets consider a typical web session of around one minute of length, where a user accesses his Facebook profile. He browses through it for 20 s, and then he follows a link to an article in a local newspaper website. After reading it, he follows a link to another article in the same newspaper website. Fig. 1 shows data captured from this example web session. The top sub-figure shows the evolution of the number of active connections during the capture and the bottom sub-figure shows the instants at which connections are opened. The session comprises three individual webpage downloads (shown as the three zones with more connection openings in the figure).

As expected, we can see that the connections from the same webpage download are very close in time. A short interval after the start of the download concentrates most of the connections. In this example, the majority of connections are persistent and, in fact, some of the connections opened for the first newspaper webpage are used in the second webpage to download new content. This effect can be checked in Fig. 2, that presents the server IP addresses to which new connections are opened through time. In the third download, only a small number of new connections are opened because previous connections are reused.

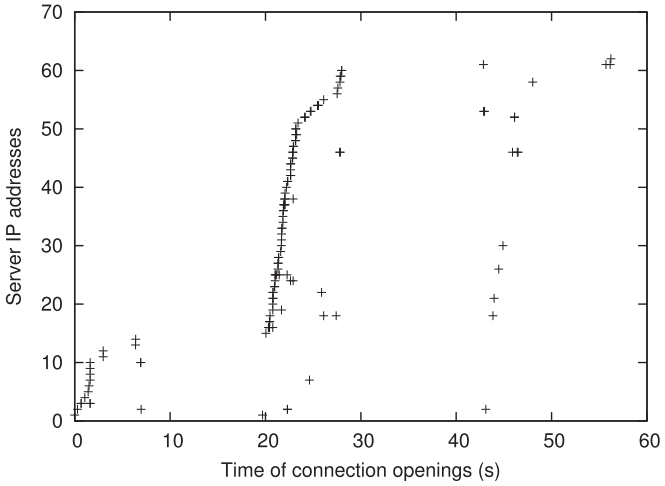

Fig. 2. Connections to different servers by IP server address, for the example web session.

At the end of the capture, the user closes the browser and the connections still open are terminated. Some of these still active connections were used to download the Facebook webpage at the beginning (they are persistent connections). As we said previously, this is one of the reasons why we do not consider end timestamps for our identification method, as they can mix connections from different webpage downloads.

An approach to identify connections belonging to a webpage download is to consider the times between consecutive connection start timestamps and decide if two consecutive connections belong to the same webpage download depending on how close they are. However, even though it is true that connections from the same webpage download are close in time to each other, it is difficult to take advantage of this fact when dealing with real web traffic. The variability in user behavior and the disruptions introduced by other applications using web ports (for example antivirus or operating system updates) generate complex traffic patterns, from which webpage downloads can be difficult to identify using simple thresholds. An improved method based on clustering can help us to get better results.

If we want to cluster connections using only their start timestamps, the problem we are facing is grouping one-dimensional data, which is a different problem than grouping multidimensional data, what typical clustering algorithms are designed for. There are some specific methods for grouping one-dimensional data. A classical one is *Jenks natural breaks optimization* (Jenks, 1967), but it requires an input parameter with the number of classes (clusters) in which the data will be divided. A generalization of this method is the K-means clustering algorithm, which has the same restriction. As we do not know beforehand how many webpages a user accesses during a period of time, we cannot use these methods. Another common approach is estimating the probability density function (p.d.f.) of connection start times. In our case, studying the shape of this density function could allow us to identify intervals with high density of connections, and therefore the instants of webpage downloads. However, estimating the p.d.f. is complex. We could use simple histograms but the information they provide can be very misleading depending on the selected box width. Complex approaches like the *Kernel density estimation* (Parzen, 1962) provide a more faithful p.d.f. but they have severe computational requirements. In any case, the exact p.d.f. cannot be estimated on the fly, preventing the system from working in real time.

We propose a modification of DBSCAN (Density Based Clustering of Applications with Noise) algorithm (Ester et al., 1996), which was specially designed to work with large databases with noise, and is nowadays one of the most popular clustering algorithms (Folino and Sabatino, 2016). Clustering algorithms like DBSCAN or K-means have been used previously for traffic classification (Erman et al., 2006) or web session clustering (Fatima et al., 2015), but not for webpage download clustering. The DBSCAN algorithm estimates the density around each data point by counting the number of points in a neighborhood and it applies a threshold to choose the points to be assigned to each cluster. Even though it was designed for multidimensional data, we have modified it in order to work with one dimension data, taking advantage of the opportunity of sorting the data. These modifications allow our algorithm (time-based density clustering) to operate in real time (i.e. clustering connections as they are captured) rather than processing a database of already captured traffic.

3.2. Time-based density clustering (TBDClust)

Similar to DBSCAN, TBDClust depends on two parameters. The first parameter is related to the distance between points, and in our case it will be a time interval between connection start times *Tbc*. The second parameter is the minimum number of points in a cluster in order to not consider the cluster noise. This second parameter will be a number of connections *Nc*. With these parameters, and considering U as the set of connections opened by a certain user (the clusters are independent for each client), we present the following definitions:

Definition 1. given two connections, x and y, and their respective start timestamps, T_x and T_y, the *distance* between those connections is defined as:

$$dist(x, y) = |T_x - T_y|$$

Definition 2. the *Tbc-neighborhood* of a connection x, denoted by $N_{Tbc}(x)$, is defined as:

$$N_{Tbc}(x) = \{y \in U | dist(x, y) \leq Tbc\}$$

Therefore, x and y are *neighbor connections* if $y \in N_{Tbc}(x)$, where *Tbc* is the maximum time distance between two neighbor connections.

Definition 3. a connection x is a *core connection* if it verifies that $|N_{Tbc}(x)| \geq Nc$. As a consequence, *Nc* represents the *minimum number of neighbors that a connection must have in order to be a core connection*.

Definition 4. a *cluster core* is a set of consecutive core connections,

$$CC = c_1, \ldots, c_n, n \geq 1 \wedge (\forall i < n, c_i \in N_{Tbc}(c_{i+1})).$$

Definition 5. a *cluster border* is the set of the neighbors of the connections in a cluster core that are not core connections,

$$CB = y \in U | (|N_{Tbc}(y)| \leq Nc) \wedge (y \in N_{Tbc}(x)) \wedge (|N_{Tbc}(x)| \geq Nc)$$

Definition 6. a *cluster* is a set of consecutive connections formed by a cluster core and its cluster border.

Definition 7. connections that do not belong to a cluster, that is, they are neither core connections nor in the neighborhood of one, are considered *noise*.

With these definitions, the clusters that we create have a core with a high density of connections and a less dense border. If for a cluster, T_{c1} is the start time of the first core connection and T_{cn} is the start time of the last one, the *total cluster length*, TCL (the time difference between the start times of the first and last connections in the cluster) has an upper limit: $TCL \leq T_{cn} - T_{c1} + 2*Tbc$. This means that the total cluster length is limited by the length of the cluster core, and thus clusters only "grow" through periods of time where the density of connections opened by the user is high.

Given the previous definitions, a connection may belong to the cluster border of two different clusters. If this happens, the algorithm will assign the connection to the oldest one. This makes sense because, as we have seen previously, the density of connections is usually higher at the beginning of a webpage download, so we expect to find more connections in the cluster border after the core than before.

Fig. 3 shows a flow diagram of the clustering algorithm. In order to explain its operation, let as consider the traffic of a particular user (client IP address), as the algorithm treats each user individually.

When a connection is opened by the user, the algorithm checks whether there are neighbors in a connection array that it keeps for each user. In this array, connections are identified by the classic 5-tuple and the timestamp of their first packet. Applying definition 2, the new connection will be a neighbor of those in the array that were opened less than *Tbc* seconds ago. If the array is empty there are no neighbors and the connection is simply added to it. If the array is not empty but none of the connections are neighbors of the new one, the array is emptied and the new connection is added to it afterwards. A neighborhood size counter is kept for the connection, with the number of neighbors it has in any given moment.

If, on the other hand, there are neighbors in the array, the connection is added to it and neighborhood size counters in the array are updated for the new connection and its neighbors. We then check the neighborhood size for the oldest neighbor of the new connection. If this neighbor is not a core connection ($|N_{Tbc}(x)| < Nc$), none of the more recent ones can be a core connection because their neighborhoods will

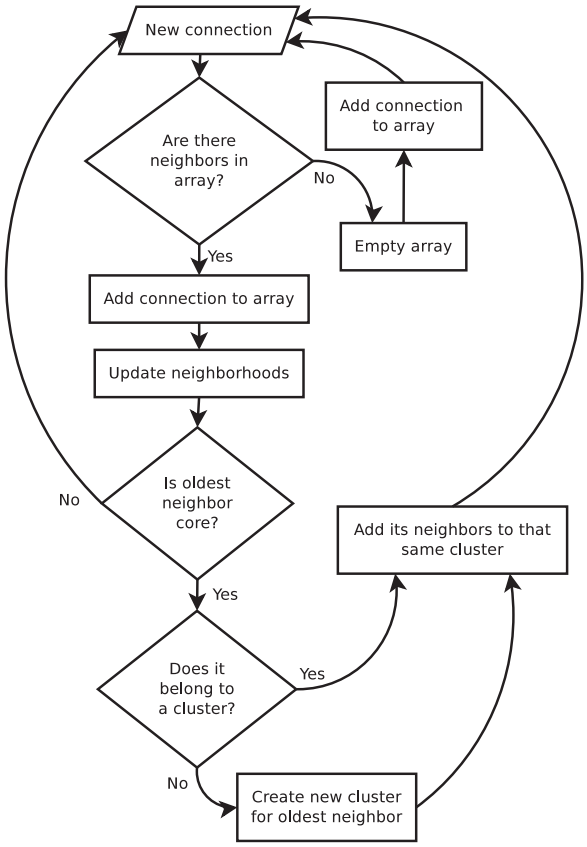

Fig. 3. Flow diagram of the clustering algorithm.

be equal or smaller. In this case, we do nothing more and wait for the next connection. However, if the oldest neighbor is a core connection ($|N_{Tbc}(x)| \geq Nc$) it has to be part of a cluster. In this case, we check whether it is already part of a cluster (it may have been a core connection or part of a cluster border before the new connection was opened). If it is part of a cluster, we make all its neighbors part of the same cluster. If it is not, we create a new cluster and make all its neighbors part of it. An exception to this occurs if a neighbor already belongs to a different cluster, in which case we leave it as it is. This is the case of border connections that could belong to two different clusters, which are assigned to the oldest one.

The array of connections that the clustering uses for each user is emptied when the new connection has no neighbors in it. This happens when the time interval between the new connection and the most recent connection in the array is bigger than *Tbc*. In this case, the new connection cannot be part of the same clusters as any of the ones in the array, and it is not necessary to keep them in memory. When the array is emptied, the connections in it that do not belong to a cluster are considered noise and discarded.

4. Results and validation

The proposed TBDClust algorithm depends on two parameters: the maximum time interval between neighbor connection starts, *Tbc*, and the minimum number of neighbors that a core connection must have, *Nc*. In this section we use different datasets to estimate the best parameter values and to validate the clustering results.

4.1. Experimental datasets

We use web traffic from two different sources: automatic captures of webpage downloads in a controlled setup (*automatic captures dataset*) and traces of real traffic from web users in a large network (*real dataset*). Both datasets are summarized in Table 1.

Table 1
Web traffic datasets.

Automatic captures dataset	20,000 downloads of the landing pages of 1000 different popular websites extracted from the top 100,000 websites of the Alexa global ranking
Real dataset	Captured web traffic from the Internet link of a network with over 15 millions of web connections during 9 workdays

The automatic captures dataset was presented in Torres et al. (2014). It is composed of 20,000 downloads with the landing pages of 1000 different popular websites. We extracted them from the top 100,000 websites of the Alexa global ranking (Alexa, 2017), taking care that the most popular and interesting sites, like Google, Facebook, or Amazon, were well represented while also collecting data from a wide variety of less popular sites from all around the world. We accessed the landing pages of these websites automatically and the browser was closed 2 min after starting the download. The network traffic was captured at least for half a minute more after this closure. This way, each individual capture file contained the download of an individual webpage, and therefore all its connections could be uniquely labeled as belonging to that individual webpage download. We used both Firefox and Google Chrome as web browsers. This automatic captures dataset will be used to test our identification system.

Testing the identification method requires also real traffic from real web users. For this purpose we have captured web traffic from the Internet link of a network with around 9.000 users during 9 workdays. This accounts for over 15 million web connections. We will refer to these captures as the real dataset. In this case the dataset has to be labeled with the connections belonging to the same individual webpage download. For this labeling, DPI and DNS name resolutions are used.

For both datasets, we work with connection records rather than packet traces. In order to obtain them we use Argus (QoSient, 2017). Argus is an open-source audit tool that is able to generate flow summaries with the same features (and more) than NetFlow/IPFIX. In particular, aside from the classic TCP/IP 5-tuple that defines each connection (IP addresses, ports and protocol) we save timestamps (start and end), total packets, total bytes and application-level bytes (upload and download). All these parameters are easily calculated and they can be inferred or at least estimated from either the downstream or upstream data if only one of the directions is available for capture from a specific vantage point.

4.2. Parameter optimization

As with any clustering algorithm, selecting an appropriate value for its parameters is key for a correct operation (Ester et al., 1996). With data as complex and variable as web traffic, we do not expect to find exact values for the parameters that offer the best results in any possible scenario. Rather than that, we would like to find ranges of parameter values for which the clustering algorithm offers good results. A more precise tuning would be advisable on a network basis (or even taking into account the characteristics of different users).

Nc is the minimum number of neighbors required for a connection to be core. Because of this, it defines minimum cluster size and it will be related to the minimum number of connections in a webpage download. In the classic DBSCAN clustering algorithm (Ester et al., 1996), a default value for *Nc* of 4 is proposed, arguing that DBSCAN behaves similarly with higher values while adding computational complexity. $Nc = 4$ produces clusters of a minimum size of 5 connections.

From the automatic captures dataset we have extracted the number of connections in each webpage download. For the 10th, 50th (median) and 90th percentiles of the number of connections per webpage

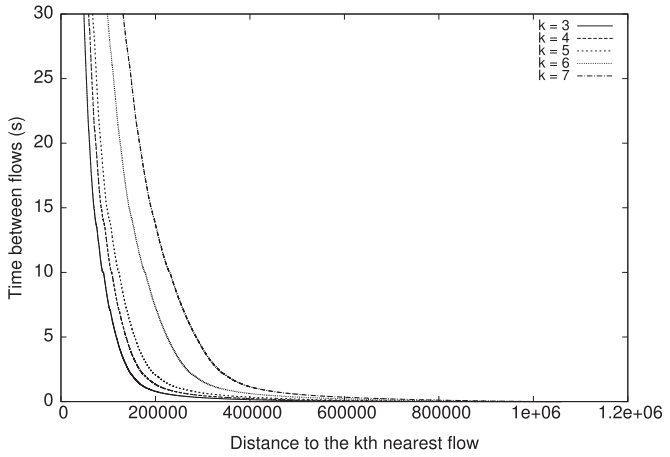

Fig. 4. k-distances graph (real dataset).

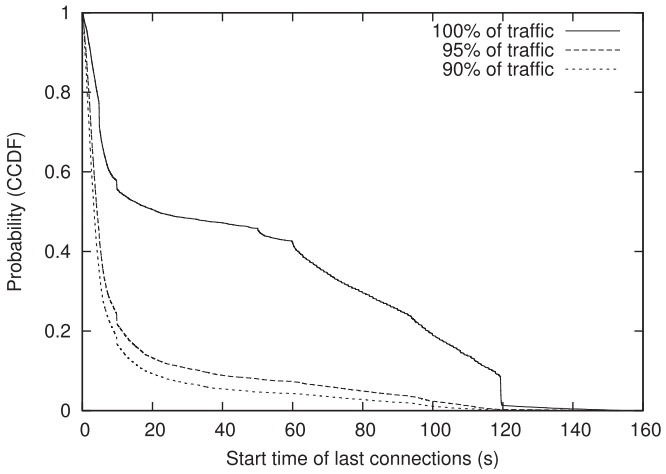

Fig. 5. Start time of last connections in webpage downloads (automatic captures dataset).

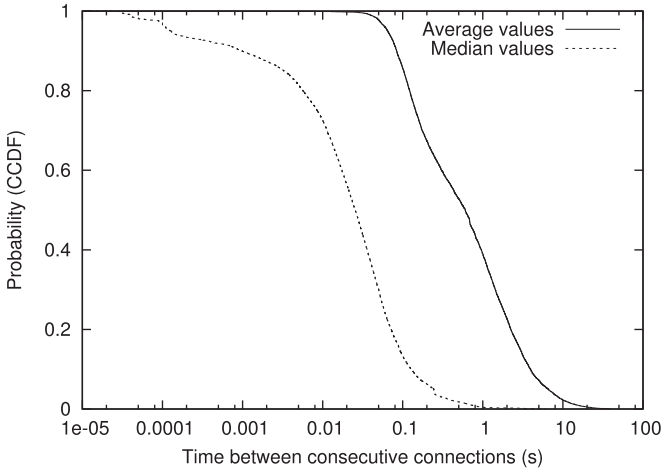

Fig. 6. Time between consecutive connections in webpage downloads (automatic captures dataset).

download, the values are 6, 43 and 125 connections respectively. This means that it is consistent to have clusters with a minimum size of 5 connections.

On the other hand, *Tbc* is the maximum time between neighbor connection start times. Its value has to be chosen carefully in order to avoid dividing a webpage download into multiple clusters while also avoiding clustering together connections from different webpage downloads. A *k-distances graph* (Ester et al., 1996) Fig. 4 can help in choosing a correct value for *Tbc*. This graph represents the time distances between connections and their k^{th} nearest neighbors. According to Ester et al. (1996), if $k = Nc$, a "knee" in the k-distances graph marks a good value for *Tbc*. This value will allow distinguishing between core and non-core connections.

For Fig. 4 we have selected a complete workday from our real dataset and calculated the k-distances (with k = 3,4,5,6,7) for every one of the over a million connections opened during the day. We have then aggregated the data from every user and sorted all the distances from highest to lowest. We set a top limit of 30 s for the ordinate axis in order to focus on the interesting part of the plot. We can see that there is not a big difference among the lines for $k \leq 5$, while the 6-distances and 7-distances lines are higher. This suggests that a top limit for *Nc* should be around 3–5, as with bigger values the smallest webpage downloads will not be considered clusters. As for *Tbc*, if we focus on $k \leq 5$, we can see a "knee" that marks that around 80% of the k-distances are smaller than 2 s.

We can check whether a *Tbc* value around 2 s makes sense by studying the connection start timestamps in the download of webpages from a variety of websites. In this case we use the automatic captures dataset. In Fig. 5 we represent the complementary cumulative distribution function (CCDF) of the start timestamps for the last connection in each webpage download. Additionally, we consider the set of connections that carry the first 90% and 95% of the traffic in the webpage downloads and represent the CCDFs of the start timestamps for the last connection in this set. This gives a better idea of the time interval during which most of the webpage is downloaded. As we can see, although connections opening late in the capture are a relatively common occurrence *most of the traffic is concentrated in the connections opened during the first seconds*. We can say that 90% of the traffic is carried by connections opened in the first 20 s for more than 90% of the captured webpage downloads (P90 = 17.85 s), although the median is in fact much lower (P50 = 3.36 s).

In Fig. 6 we look at connection start times from a different perspective by considering time differences between consecutive connection start timestamps. In that figure we have calculated the sample average and median values of the time differences between consecutive connection starts in each capture of the automatic captures dataset and represented the CCDF of both statistics. The CCDF of the sample median shows that connections of the same webpage download are usually very close from each other. Some time-differences are bigger, as evidenced by the CCDF of the sample average which is far less robust against extreme values. From this figure we obtain that 90% of the sample values of median and average are below 0.13 and 3.95 respectively. These values are small enough if we compare them with the time a user spends browsing in a webpage.

Webpage dwell times are difficult to model as they depend on the user's navigation habits, the interest and complexity of the webpage and its actual content. For example, a user will take some time in reading a news article or watching a video but may follow a link to another webpage rapidly after using a search engine. A minimum dwell time of 30 s has often been used for webpages that are of interest to the user (Kim et al., 2014) although very simple or uninteresting webpages may experience lower dwell times.

We can describe a typical webpage download as a group of connections that, if considered as a whole, span a limited time frame and that, if considered in consecutive pairs, happen very close to each other. We have quantified the download length and the time between consecutive connections by calculating the 90th percentile from our automatic captures dataset using medians. The results are 17.85 s for download length and 0.13 s for time between consecutive connections.

In brief, for *Nc* we will follow the recommendations for tuning the algorithm and use $Nc = 4$. This value is consistent with our experimental knowledge of the problem and the k-distances graph suggests that choosing $Nc = 3$ or $Nc = 5$ would not produce very different

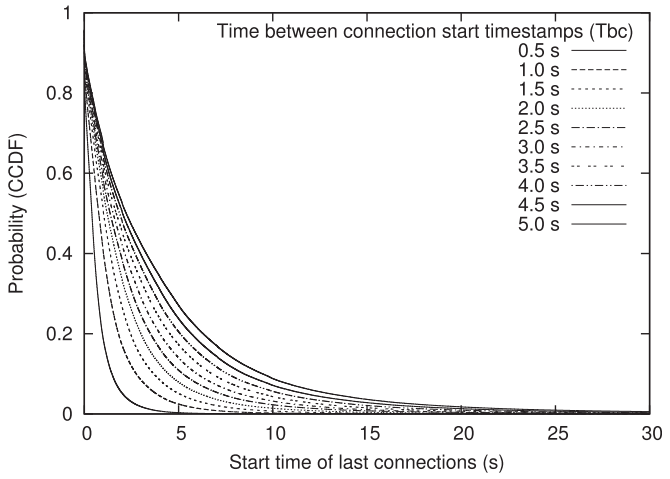

Fig. 7. TBDClust: CCDFs of cluster length, for different Tbc values (real dataset).

results. Choosing an exact value for *Tbc* is more complicated, although we have seen that it should be in the vicinity of 2 s. This seems consistent with Fig. 6, as most connections from the same webpage download are closer than 2 s to each other.

4.3. TBDClust validation using a one-day real dataset

In this section we validate the working ranges for the *Nc* and *Tbc* parameters. We work with data from one whole workday from the real dataset. We will later on test the final tuned system with the full dataset and check whether it still operates correctly.

First we have analyzed the dependency with the *Tbc* parameter. We have run the TBDClust algorithm with the real dataset, obtaining the results shown in Figs. 7 and 8 for different values of *Tbc*. Fig. 7 shows the CCDF of cluster length considered as the time difference from the first to the last connection in the cluster. Fig. 8 shows the CCDF of the number of connections.

The cluster length is influenced very little by the chosen threshold *Tbc* (Fig. 7). This makes sense because the noise connections will not be taken into account unless they are very close to the connections of the webpage download. In fact, it is less problematic if the first connection of a cluster is a "noise connection" as there is no time limit from the start of this connection and the end of the cluster, and webpage downloads will not be split because of this reason. Nevertheless, even if there is not a maximum cluster length, this method usually avoids creating massive clusters for very active users because clusters only grow through core connections that have a minimum number of close neighbors.

On the other hand, as *Nc* is the minimum number of connections for a cluster, the method does not create very small clusters (less than 5 connections for $Nc = 4$). This is good because smaller clusters would not contain a full webpage download. These small clusters can contain web connections corresponding to updates from the antivirus, operating system or any other automatic task. Using a small value of *Nc*, the algorithm will consider these clusters as noise.

We analyze the results for different values of *Tbc*. Fig. 9 shows the percentage of connections that have been included in the correct clusters. From all the connections that we have labeled as belonging to a certain webpage download, this measures how many are really in the cluster associated to that webpage download. The *Tbc* parameter is shown in the bottom abscissa axis. The values of the percentage of clustered connections are in the 78–90% range, depending on the *Tbc* values. This has to be interpreted taking into account that we are working in a very noisy environment where a sizable percentage of the connections to ports 80 and 443 belong to applications other than web browsing (such as application updates, as mentioned previously). The curve tends to stabilize as the generated clusters start to group almost all the connections from each download. This suggests that the range for the parameter *Tbc* has been properly selected.

Fig. 10 shows the average length of those clusters, measured as the time difference between the first and the last connection in the cluster. The increase in cluster time length is related to the parameter *Tbc*. It tends to stabilize on values of *Tbc* around 5 s. This behavior is thanks to its ability to accurately find time intervals with a high density of connections. Connections considered as noise are not included in any cluster (hence in no webpage download).

Previous Figs. 9 and 10 do not allow us to check how well formed the resulting clusters are. We need to perform a validation process that checks whether each cluster comprehends a full webpage download. We use two metrics from the clustering literature (Ballou and Pazer, 2003; Khaleghi et al., 2016): consistency and completeness.

We define the cluster *consistency* as the percentage of the connections in the cluster that are related to the same webpage download. Two connections would be related if they belong to the same webpage download and unrelated if they are not. Using this criterion, a consistency of 100% will mean that all the connections inside a cluster are related and belong to the same webpage download.

Fig. 11 shows the average cluster consistency obtained with the TBDClust algorithm, depending on the parameter *Tbc*. The data comes from the real dataset. As we can see, for $Tbc \leq 2\ s$, the method reaches consistency averages very near to the expected maximum. This means that the method is not prone to mixing connections of different

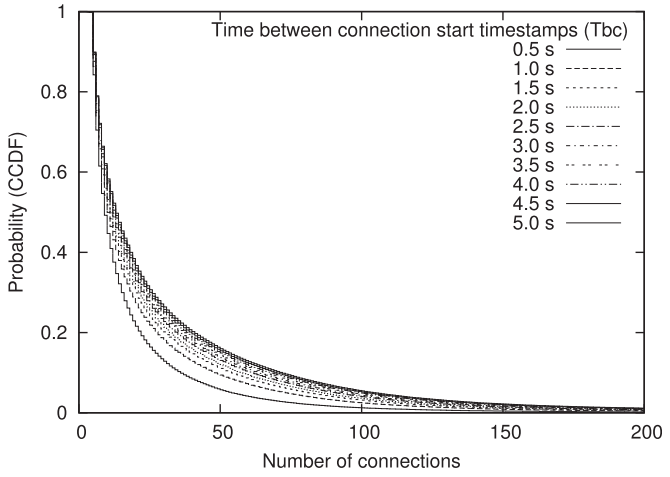

Fig. 8. TBDClust: CCDFs of the number of connections per cluster, for different Tbc values (real dataset).

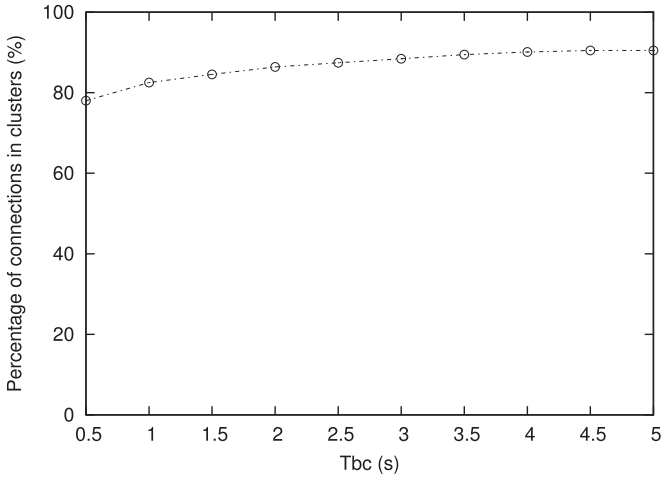

Fig. 9. Percentage of connections (in clusters of more than 5 connections) (real dataset).

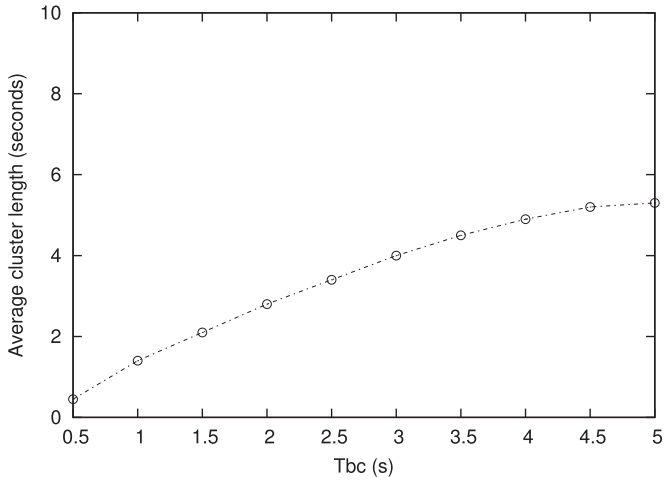

Fig. 10. Average cluster length (real dataset).

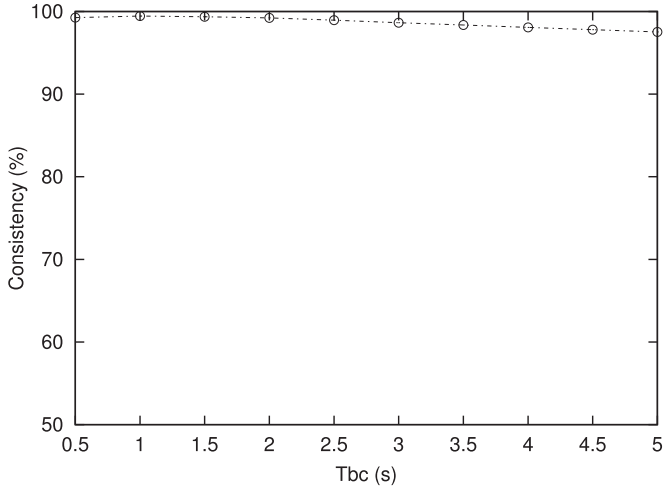

Fig. 11. Consistency (real dataset).

webpage downloads. A larger value of *Tbc* would not enlarge a cluster as there would not be enough connections to become core. The occurrence of a new set of connections would imply the creation of a new cluster and not the growth of the previous one.

Internal consistency is not enough to validate the clustering method. We also need to ascertain that the connections that belong to the same webpage download are not divided into multiple clusters. We define the *completeness* of a cluster as the percentage of connections from the original webpage download that are classified in the same cluster. Fig. 12 shows the completeness for the real dataset. Its average stabilizes in values over 90% when the parameter *Tbc* takes values larger than 2 s. The lack of perfect completeness is caused by some asynchronous connections opened by the web browser very late from the initial webpage download and therefore it is arguable whether the last connection should belong to the same cluster. Another cause for this difference are websites that use gallery-like webpages, in which the user cycles through different content (for our purposes, different webpages) but in the labeling they are considered as the same webpage download.

The fact that the completeness stabilizes is important because it shows that if we increase the parameter *Tbc* (aggregating more connections to the clusters) most of these new connections in the cluster will not belong to the same webpage download as consistency falls but completeness stays almost constant. Although the presented values for completeness are not completely satisfactory (slightly above 90%) they are not a critical factor on webpage classification. Even if the

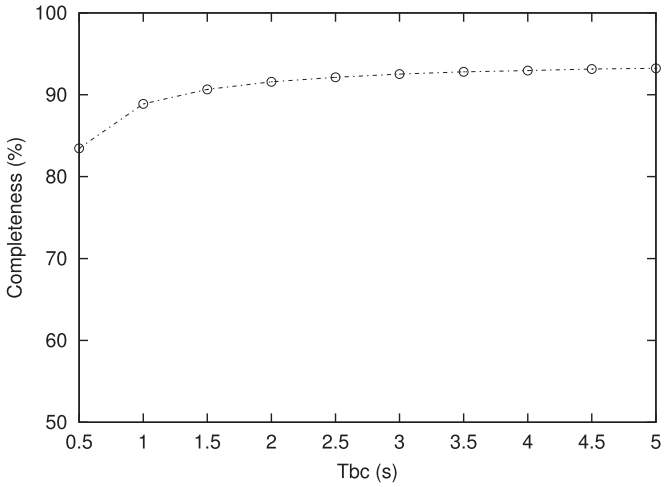

Fig. 12. Completeness (real dataset).

webpage download is not contained in a single cluster, the second cluster could be correctly classified if its first connection targets the white/black-listed server. Therefore, these are worst-case results, where we consider the second cluster as never correctly classified.

As for the optimal value of the *Tbc* interval, studying Fig. 11 we see that consistency values are more or less stable up to 2 s; from that point, they start to decrease. On the other hand, completeness grows while we increase *Tbc* up to 2 s and then it changes much more slowly. Because of this, a value of 2 s is a good value for *Tbc* for the selected workday with the real dataset. In the following section we will use the full 9-days real dataset to test the algorithm with ($Nc = 4$; $Tbc = 2\ s$) and check whether consistency and completeness values remain in acceptable values or not.

4.4. TBDClust validation using a one-week real dataset

We have tested TBDClust with the parameters ($Nc = 4$; $Tbc = 2\ s$) for the rest of the days in the real dataset. The average daily values for consistency and completeness are shown in Figs. 13 and 14. Both parameters remain close to the ones obtained for the day 1, for which the algorithm was tuned (shown in the figure in a lighter shade). This shows that once tuned with a sample of network traffic, the time-based density clustering algorithm offers a stable performance.

One of our concerns when designing the clustering system was that modern users are accustomed to tab based browsing, which allows them to browse through different websites concurrently. We thought

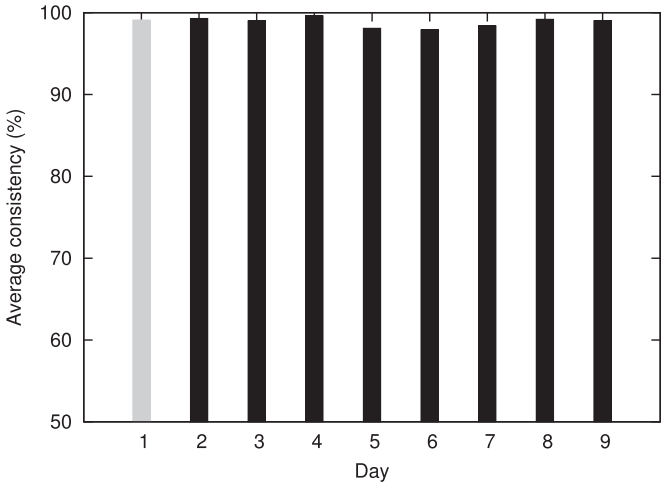

Fig. 13. Testing TBDClust, daily average consistency (real dataset).

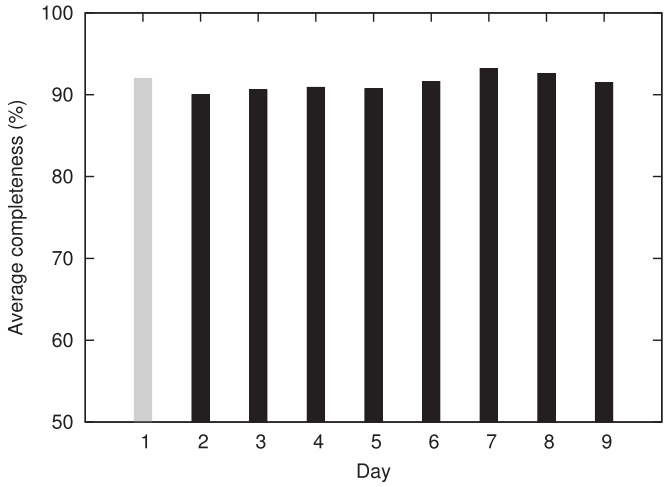

Fig. 14. Testing TBDClust, daily average completeness (real dataset).

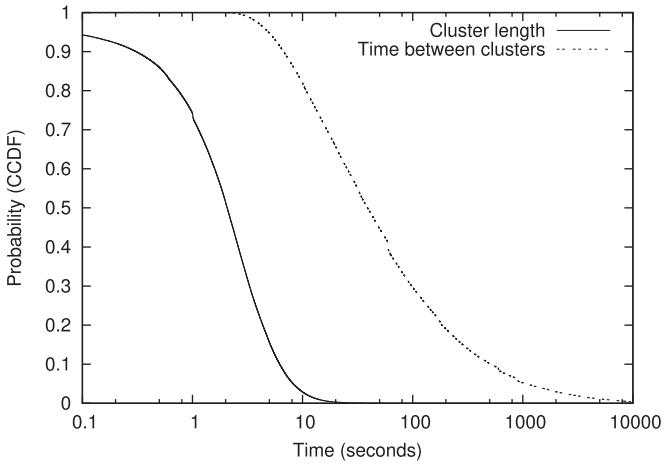

Fig. 16. Testing TBDClust, CCDFs of the cluster length and time between consecutive cluster starts (real dataset).

that this might cause a time-based method to mix simultaneous downloads of different webpages. However, this does not seem to be problematic, given the fact that consistency values remain constant in Fig. 13 and that they are close to the expected value calculated with the automatic captures dataset. Even though users keep multiple tabs open and browse through different websites, the time intervals between the download of each individual webpage are usually long enough, allowing the clustering to work properly.

We present some additional characteristics that describe the clusters created by the algorithm, using the real dataset. Fig. 15 shows the CCDF of the daily number of clusters per user. With a median of 59 clusters and 10th and 90th percentiles at 18 and 179 clusters respectively, the number of clusters is in a reasonable range, considering that it should be closely related to the number of webpages visited daily by each user. For comparison, the authors from Weinreich et al. (2008) found that the average number of webpages visited per user and day ranged from 25 to 284.

Although rare, some of our users have a very high daily number of clusters (P99=426). We have checked these cases manually and we have found that most of the clusters are related to abnormal cases. In some cases, webpages with automatic reload are left open in the browsers, so every some minutes during the day they are reloaded. In another case, a malfunctioning web browser was attempting to download favicons over and over. We have also checked that for non web applications using web ports (p.e. antivirus updates), TBDClust filters most of this traffic as noise.

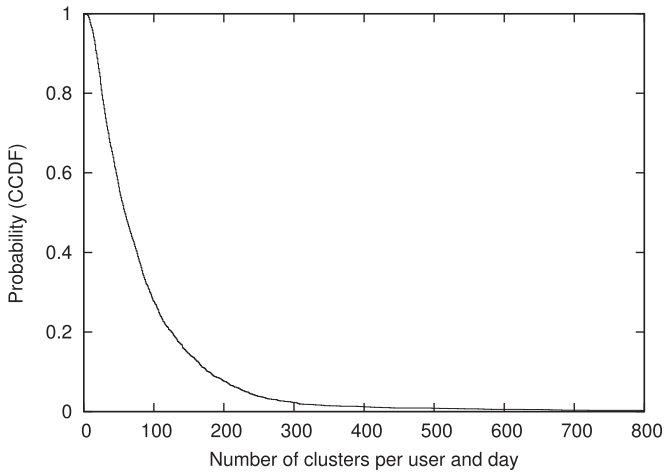

Fig. 15. Testing TBDClust, CCDFs of the number of clusters per user and day (real dataset).

Fig. 16 shows the distributions of cluster length in seconds and the time between consecutive cluster start timestamps. Values for both parameters are close to what we expected. With a median length of 2.1 s, our clusters are shorter than the webpage downloads from the automatic captures dataset. Nevertheless, this is understandable because the webpages visited in the automatic captures dataset were only landing pages (which usually are more complex) and because in real traffic some of the content may be already cached in the host as users usually revisit the same webpages. Median time between clusters is 39 s which seems a reasonable webpage dwell time.

Another interesting consideration arises if we consider extreme percentiles for both distributions. The 90th percentile of cluster length is 6.2 s while the 10th percentile of the time between consecutive cluster start times is 6.7 s. This shows that even the longest clusters are shorter than the time between the closest ones, which makes the possibility of overlapping two different webpage downloads in the same cluster a remote one.

5. TBDClust performance evaluation

An important design objective in TBDClust was the capability to be used in real time. We want the clustering to take place as the connections arrive to a monitoring system, network firewall or any other middlebox. The presented algorithm allows for this behavior. In this section we show that its complexity is so low that it does not limit its usability in high traffic load scenarios.

As described in Section 3, the algorithm takes two passes through arrays for each new connection arrival. The array depth depends on the number of neighbor connections and therefore it is variable with time. In order to evaluate the running time of the algorithm we prepared a high traffic load scenario. We show the results for a whole day of web traffic, consisting of 800.4 GiB from 816.7 million packets in 6.3 million TCP connections. We replayed this traffic trace at the maximum possible speed and measured how fast the algorithm implementation consumed the traffic. We used a Commodity Off-The-Shelf (COTS) server with an Intel Xeon E5-2609 CPU running at 1.7 GHz. We ran the algorithm using a single CPU core or splitting the traffic between two cores operating in parallel.

Fig. 17 shows the processing, averaged per second. In order to compare the runs using 1 and 2 cores we plot the speed versus the traffic trace length and not versus the real time it took to process the trace (which was different, shorter for the faster scenario). The experiment using a single CPU core was capable of consuming 10 Gbps for 97.3% of the time, while using 2 CPU cores the computer was always capable of consuming 10 Gbps of network traffic and 97.8% of the time it could consume 20 Gbps.

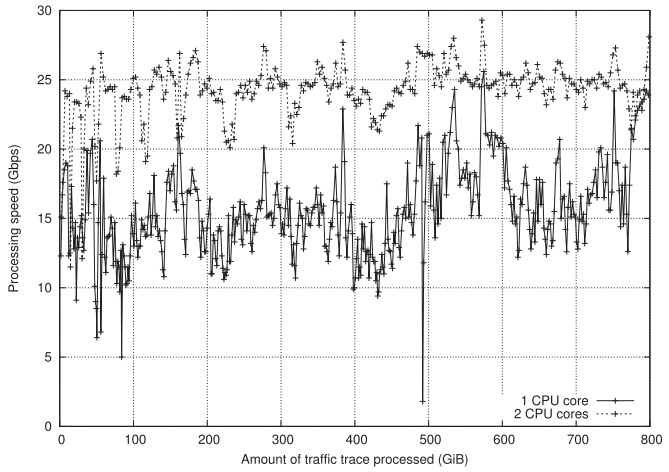

Fig. 17. Testing TBDClust running time.

Any tool for high-speed TCP traffic processing must decode packet headers, sort TCP connections into lists or arrays, manage memory, etc. The presented time-based clustering algorithm adds a small amount of work, but only for connection start times, and not for every other packet, therefore processing speed is not critically limited by TBDClust.

In comparison, DPI based clustering could achieve close to 100% completeness and consistency but only in very limited scenarios because it would only work with non-encrypted traffic, which in the public Internet is less abundant every day (Finley, 2014). DPI requires processing every data packet. It has to reconstruct the whole TCP stream, in order to access the HTTP data, parse the HTML content and locate the embedded URLs. Based on those URLs it could recognise later HTTP sessions corresponding to the same webpage. However, in modern webpages some URLs are constructed on-the-fly by Javascript code. DPI could not detect those URLs in the downloaded HTML or Javascript code as they are constructed depending on user actions. Therefore, nowadays not even DPI could reach 100% in every clustering metric and for every webpage, while its performance is clearly handicapped by its computing complexity and high memory requirements.

6. Conclusions

In the case of a mobile Internet provider that offers a pay-per-use data plan, sometimes the provider wants to allow free browsing through strategic commercial web sites or the authorities require to allow browsing of institutional or emergency websites at no cost. The most common way to allow this differentiation has been to have a white list of domains or IP addresses to which the users are allowed access. In this paper we have proposed having a white list only with the main domain name or main IP address of the website that we want to allow, and use a clustering method to identify those TCP connections related to the main connection. This will allow the user to load contents from all the necessary servers, keeping a good browsing experience.

The proposed clustering method is based on a time density scheme that is able to group TCP connections that belong to the same webpage download. It is an adaptation of the well-known DBSCAN clustering technique over one-dimensional data. The method does not need deep packet inspection. It needs only very basic information from the first packet of each TCP connection: timestamp and client IP address. Therefore, the proposed method is simple and fast, which makes it suitable for real-time operation. We have proved that a single CPU core in commodity hardware is capable of processing 10 Gbps of network traffic.

The TBDClust (Time-Based Density clustering) algorithm has been tested using both automatically collected samples of webpage downloads and real web traffic from a network with more than 9.000 users. We have been able to obtain clusters with average internal consistency of around 99% and average completeness of near 92%. This means a very low rate of error in connection identification to its corresponding webpage download. These results have been shown to be stable over time. This allows the operation in almost any network with very low tuning.

Acknowledgements

This work is supported by Spanish MINECO through project PIT (TEC2015-69417-C2-2-R).

References

Alexa, Toolbar and Site Rankings. ⟨http://www.alexa.com⟩ (last checked: Feb. 7, 2017).
Ballou, D.P., Pazer, H.L., 2003. Modeling completeness versus consistency tradeoffs in information decision contexts. IEEE Trans. Knowl. Data Eng. 15 (1), 240–243.
Belshe, M., Peon, R., Thomson, M., Oct. 2014. Hypertext Transfer Protocol version 2. ⟨https://datatracker.ietf.org/doc/draft-ietf-httpbis-http2⟩.
Bianco, A., Mardente, G., Mellia, M., Munafo, M., Muscariello, L., 2009. Web user-session inference by means of clustering techniques. IEEE/ACM Trans. Netw. 17 (2), 405–416.
Butkiewicz, M., Madhyastha, H.V., Sekar, V., 2011. Understanding website complexity: measurements, metrics, and implications. In: Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement, IMC '11, ACM, New York, NY, USA, pp. 313–328.
Butkiewicz, M., Madhyastha, H.V., Sekar, V., 2014. Characterizing web page complexity and its impact. IEEE/ACM Trans. Netw. 22 (3), 943–956.
Callado, A., Kelner, J., Sadok, D., Kamienski, C.A., Fernandes, S., 2010. Better network traffic identification through the independent combination of techniques. J. Netw. Comput. Appl. 33 (4), 433–446.
Catledge, L.D., Pitkow, J.E., 1995. Characterizing browsing strategies in the world-wide web. Comput. Netw. ISDN Syst. 27, 1065–1073.
Charzinski, J., 2010. Traffic properties, client side cachability and CDN usage of popular web sites. In: Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance, Vol. 5987 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, pp. 136–150.
Claise, E.B., Jan. 2008. RFC 5101: Specification of the IPFIX Protocol for the Exchange of IP Traffic Flow Information.
Erman, J., Arlitt, M., Mahanti, A., 2006. Traffic classification using clustering algorithms. In: Proceedings of the SIGCOMM workshop on Mining network data (MineNet '06), pp. 281–286.
Ester, M., Kriegel, H., Sander, J., Xu, X., 1996. A density-based algorithm for discovering clusters in large spatial databases with noise, In: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, KDD-96, AAAI Press, pp. 226–231.
Fang, C., Liu, J., Lei, Z., 2016. Fine-grained HTTP web traffic analysis based on large-scale mobile datasets. IEEE Access. 4, 4364–4373.
Fatima, B., Ramzan, H., Asghar, S., 2015. Session identification techniques used in web usage mining: a systematic mapping of scholarly literature. Online Inf. Rev. 40 (7), 1033–1053.
Finley, K., May 2014. Encrypted Web Traffic More than Doubles after NSA Revelations. ⟨http://www.wired.com/2014/05/sandvine-report/⟩.
Folino, G., Sabatino, P., 2016. Ensemble based collaborative and distributed intrusion detection systems: a survey. J. Netw. Comput. Appl. 66, 1–16.
Fortino, G., Mastroianni, C., 2009. Next generation content networks. J. Netw. Comput. Appl. 32 (5), 941–942, (next Generation Content Networks).
Free Facebook browsing in Vodafone Mobile Operator at Albania, ⟨http://www.vodafone.al/vodafone/Facebook_3837_2.php⟩ (last checked: Feb. 7, 2017).
Google Safe Browsing for developers, ⟨https://developers.google.com/safe-browsing⟩ (last checked: Feb. 7, 2017).
Ihm, S., Pai, V.S., 2011. Towards understanding modern web traffic. In: Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference, IMC '11, ACM, New York, NY, USA, pp. 295–312.
Jenks, G., 1967. The data model concept in statistical mapping. In: International Yearbook of Cartography, Vol. 7, George Philip, pp. 186–190.
Khaleghi, A., Ryabko, D., Mary, J., Preux, P., 2016. Consistent algorithms for clustering time series. J. Mach. Learn. Res. 17 (1), 94–125.
Khandelwal, H., Hao, F., Mukherjee, S., Kompella, R., Lakshman, T., 2013. Cobweb: In-network cobbling of web traffic. In: Proceedings of IFIP Networking Conference, pp. 1–9.
Kim, Y., Hassan, A., White, R.W., Zitouni, I., 2014. Modeling dwell time to predict click-level satisfaction. In: Proceedings of the 7th ACM International Conference on Web Search and Data Mining, WSDM '14, ACM, New York, NY, USA, pp. 193–202.
Liu, H., Kešelj, V., 2007. Combined mining of web server logs and web contents for classifying user navigation patterns and predicting users' future requests. Data Knowl. Eng. 61 (2), 304–330.
Lu, C.-N., Huang, C.-Y., Lin, Y.-D., Lai, Y.-C., 2016. High performance traffic classification based on message size sequence and distribution. J. Netw. Comput. Appl. 76, 60–74.
Macia-Fernandez, G., Wang, Y., Rodríguez-Gómez, R.A., Kuzmanovic, A., 2012.

Extracting user web browsing patterns from non-content network traces: the online advertising case study. Comput. Netw. 56 (2), 598–614.
Nguyen, T.T.T., Armitage, G., 2008. A survey of techniques for Internet traffic classification using machine learning. Commun. Surv. Tutor., IEEE 10 (4), 56–76.
Parzen, E., 1962. On estimation of a probability density function and mode. Ann. Math. Stat. 33 (3), 1065–1076.
QoSient, Argus: Audit Record Generation and Usage System, ⟨http://www.qosient.com/argus/⟩ (last checked: Feb. 7, 2017).
Schatzmann, D., Mühlbauer, W., Spyropoulos, T., Dimitropoulos, X., 2010. Digging into HTTPS: flow-based classification of webmail traffic. In: Proceedings of the 10th Conference on Internet Measurement (IMC '10), ACM, New York, NY, USA, pp. 322–327.
Schneider, F., Agarwal, S., Alpcan, T., Feldmann, A., 2008. The new web: Characterizing AJAX traffic. In: Passive and Active Network Measurement, vol. 4979 of Lecture Notes in Computer Science, Springer, pp. 31–40.
Torres, L.M., Magaña, E., Izal, M., Morato, D., 2014. Characterizing webpage load from the perspective of TCP connections. In: Proceedings of the 2014 Federated Conference on Computer Science and Information Systems, vol. 2 of Annals of Computer Science and Information Systems, IEEE, pp. 977–984.
Weinreich, H., Obendorf, H., Herder, E., Mayer, M., 2008. Not quite the average: an empirical study of web use. ACM Trans. Web 2 (1), 1–31.
Xie, G., Iliofotou, M., Karagiannis, T., Faloutsos, M., Jin, Y., 2013. Resurf: Reconstructing web-surfing activity from network traffic. In: Proceedings of IFIP Networking Conference, pp. 1–9.
Zhang, J., Chen, X., Xiang, Y., Zhou, W., Wu, J., 2015. Robust network traffic classification. IEEE/ACM Trans. Netw. 23 (4), 1257–1270.
Zink, M., Suh, K., Gu, Y., Kurose, J., 2009. Characteristics of YouTube network traffic at a campus network: measurements, models, and implications. Comput. Netw. 53 (4), 501–514.

Luis Miguel Torres received his degree in Telecommunications Engineering by the Public University of Navarre (UPNA) in 2008 and his MSc and Ph.D. in Communications Technology by the same university in 2009 and 2015 respectively. From 2008–2016 he has worked with the Networks, Systems and Services research group at the department of Automatics and Computation at the Public University of Navarre. Nowadays he works as a network traffic analyst in Naudit HPCN. His professional activity centers around the analysis of Internet traffic.

Eduardo Magaña received his M.Sc. and Ph.D. degrees in Telecommunications Engineering from Public University of Navarra, Pamplona, Spain, in 1998 and 2001, respectively. He is an associate professor at Public University of Navarra. During 2002 he was a postdoctoral visiting research fellow at the Department of Electrical Engineering and Computer Science, University of California, Berkeley. His main research interests are network monitoring, traffic analysis and performance evaluation of communication networks.

Daniel Morató received the M.Sc. degree in telecommunication engineering and the Ph.D. degree from the Public University of Navarre, Spain. During 2002 he was a visting postdoctoral fellow at the Electrical Engineering and Computer Sciences Department, University of California, Berkeley. Since 2006 he has been working at the Department of Automatics and Computer Sciences, Public University of Navarre, as an associate professor. His reseach interests include high-speed networks, performance and traffic analysis of Internet services and network monitoring.

Santiago Garcia-Jimenez received his M.Sc. and Ph.D. degrees in Computer Science from Public University of Navarra, Pamplona, Spain, in 2007 and 2013, respectively. He had a fellowship in Telecommunications Engineering Department in 2003–2008 and he has participated in several European projects. His interests are network monitoring based on active and passive measurements, network measurement platforms, discovery of Internet topology, IP addresses alias resolution and web troubleshooting.

Mikel Izal received his M.Sc. and Ph.D. degrees in telecommunication engineering in 1997 and 2002 respectively. In 2003 he worked as a scientific visitant at Institute Eurecom, Sophia-Antipolis, France, performing measures in network tomography and peer-to-peer systems. Since then, he has been with the Department ofAutomatics and Computer Sciences of the Universidad Pblica de Navarra where he is an Associate Professor. His research interests include traffic analysis, network tomography, high speed next generation networks and peer to peer systems.