Request Dependency Graph: A Model for Web Usage Mining in Large-scale Web of Things

---

Citation:
Jun Liu, Cheng Fang, and Nirwan Ansari, " Request Dependency Graph: A Model for Web Usage Mining in Large-scale Web of Things," *IEEE Internet of Things Journal*, DOI: 10.1109/JIOT.2015.2452964, vol. 3, no. 4, pp. 598-608, Aug. 2016.

URL:
http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7150334

Request Dependency Graph: A Model for Web Usage Mining in Large-scale Web of Things

Jun Liu, Cheng Fang
School of Information and Communication Engineering
Beijing University of Posts and Telecommunications
Beijing, China, 100876.
Email: liujun, fone@bupt.edu.cn

Nirwan Ansari
Electrical and Computer Engineering Department
New Jersey Institute of Technology
New Jersey, US, 07102.
Email: nirwan.ansari@njit.edu

***Abstract*—In the Web of Things environment, web traffic logs contain valuable information of how people interact with smart devices and web servers. Mining the wealth of information available in the web access logs has theoretical and practical significance for many important applications like network optimization and security management. The first critical step of the mining task is modeling the relationships among HTTP requests for accessing web objects to investigate the behavior of web clients. In this paper, we introduce the *request dependency graph*, a graph representation of the relationships among HTTP requests. Conceptually, a directed link from A to B in the graph means that the accessing of web object B is caused by the accessing of A, i.e., B depends on A. We propose a methodology to establish such a graph by mining the temporal and causal information among aggregated HTTP requests. To demonstrate the value and effectiveness of the proposed model, we design and implement an algorithm for primary requests identification, which is a critical task of web usage mining, based on the request dependency graph. Evaluation results from a large-scale real-world web access log shows that the request dependency graph is a useful tool for web usage mining.**

I. Introduction

To create innovative and user-friendly services on top of the Internet of Things [1], the Web of Things (WoT) [2] brings an integrated architecture for the convergence of data from smart devices into web applications. In the WoT environment, users can leverage convenient web mechanisms like browsing, searching and linking to interact with smart devices and web servers. Therefore, mining the web traffic in WoT is increasingly critical for network administrators for operational and security purposes. For example, understanding traffic characteristics is important for managing and provisioning WoT network resources, and monitoring activities of devices is important to identify anomalies and prevent attacks.

According to the locations of acquired traffic logs, web traffic mining can be categorized into three types: client-side, server-side, and network-side traffic mining. Mining the web traffic at the client-side and server-side has been an active area of research in recent years. However, mining network-side web traffic logs, which are composed by massive HTTP requests with information of corresponding responses, poses many technical challenges that arise from the large volume and low quality of data [3]. The traditional emphasis of web traffic analysis at network-side has been based on the statistical and structural properties of the overall web environment [4][5][6]. There are relatively few studies that consider the HTTP (the standard protocol underlying the web) requests as a set of associated records to extract the interests and the preferences of individuals at the network-side.

In this paper, we introduce the concept of the *Request Dependency Graph* (RDG), which models the dependency relationships among HTTP requests to analyze the behavioral characteristics of web traffic, such as interaction structures of web objects and browsing patterns of web clients. Given a web traffic log, the nodes of the request dependency graph are all the accessed web objects in the log. Each node is characterized with a strength, which is the occurrence count of the accessed object. A directed edge from a predecessor node A to a successor node B represents that the access of B is caused by the access of A, referred to as the dependency relationship. The weight of an edge is the number of occurrences of the dependency relationship between the two nodes appeared in the log. The request dependency graph can be established by a learning algorithm over a web traffic log. Based on the study of a request dependency graph derived from real network data, we find that the request dependency graph is large, sparse and seemingly complex. It exhibits power-law characteristics in the distributions of a number of graph properties, such as degree and weight distributions.

To demonstrate the value of the request dependency graph, we apply it on an important preprocessing task for web usage mining, e.g., primary HTTP requests identification. It is to identify the initial HTTP requests of opening a web page, which is triggered by a user click or a device access action, from the captured web traffic logs. We use the request dependency graph model to describe the complex web browsing behavior. Based on the graph, we propose a two-step algorithm to identify the primary requests from a huge number of HTTP requests of web pages and embedded objects. The first step is to establish the request dependency graph from captured HTTP requests. The second step is to identify the primary requests by a statistical inference approach. The primary requests are identified by comparing their probabilities of being the primary request with a self learned threshold. Experiments on real-world data demonstrate that our method can achieve higher accuracy in comparison with traditional data cleaning method.

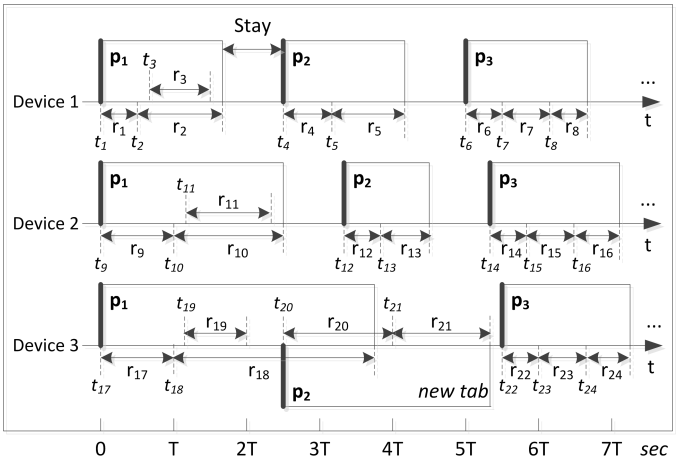


Fig. 1. Web browsing behavior

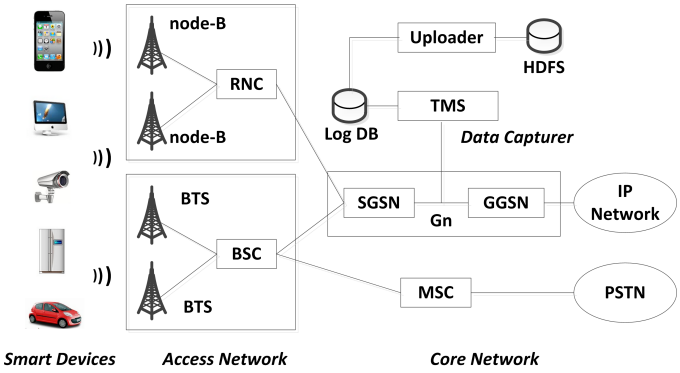


Fig. 2. System architecture for capturing traffic at the network-side

The main contributions of this paper are in three aspects: (i) we introduce the request dependency graph model to describe the web browsing behavior and provide a methodology to construct such a graph based on mining web traffic logs; (ii) we demonstrate the value of the request dependency graph by proposing a new effective primary requests identification algorithm; (iii) we conduct extensive experiments based on real-world datasets to illustrate the structural characteristics of the graph and validate the primary requests identification algorithm. The idea of the dependency graph was preliminarily presented at WOCC 2014 [7], and this paper is an elaborated and extended version, which includes, in particular, the investigation of structural characteristics of the proposed graph model and enhanced evaluation of the primary requests identification algorithm.

The rest of the paper is organized as follows. In Section II, we describe the basic concepts of web browsing behaviors. In Section III, we introduce the request dependency graph model and the methodology to establish the graph. In this section, we also reveal the structural characteristics of the request dependency graph based on a dataset collected from a large cellular network. Then, in Section IV, we propose a novel primary requests identification algorithm based on the request dependency graph model and evaluate the effectiveness of the algorithm by conducting a set of experiments using ground truth and real-world datasets. In Section V, we review the related works of our research. Finally, we present the future work and conclude the paper in Section VI.

II. Basic Concepts of Web Browsing

At first, we look into interactions between smart devices and web servers reflected in the network. Fig. 1 depicts a sample web browsing behavior, in which time flows from left to right. For simplicity, we only illustrate the process of three smart devices (*Device* $1-3$) accessing three pages (p_i) of a web server. Basically, the web browsing behaviors are related to two factors: the structure of the web server and the way the devices interacted with the web server. In general, a web server comprises two types of elements: web pages and embedded objects in each page. All pages and embedded objects are identified by Universal Resource Locators (URLs), which are represented as hyperlinks in the web pages. For example, requests r_1 and r_9 in Fig. 1 have the same URL because they are produced by accessing hyperlinks pointing to the same web page p_1. The structure of a web server is determined by these hyperlinks among the web pages. When a device accesses a hyperlink to open a page, the web client like a web browser or an embedded web application will send an initial HTTP request containing the URL of this page to the web server. Responded page content of this initial request usually contains many hyperlinks of the embedded objects. After parsing these hyperlinks, the web client on the device produces a set of requests to retrieve embedded objects from web servers in a multi-thread manner. During the period of loading a web page, devices may instill different browsing behaviors. In terms of how devices interact with the websites, most devices usually wait till a full page loaded, and stay for a while before open another hyperlink in the page (*Device* 1 and 2). Some impatient devices, however, branch out to other pages in parallel threads, such as a user access multiple web pages at the same time in multiple tabs of the web browser. This will result in some overlap request sequences of multiple pages, such as pages p_1 and p_2 opened by *Device* 3.

Although the cause of a small piece of HTTP requests can be explained by such a simple browsing process based on human perception, the requests generated by a huge number of devices that compose the web traffic logs used by network operators for web usage mining are mixed and complicated at the network-side. Fig. 2 illustrates a network environment example to capture web traffic logs at the network-side in a cellular WoT network. The example network serving both 2G and 3G devices consists of three major parts: smart devices, the access network and the core network. A smart device communicates with a cell tower in the access network which forwards its data service traffic to a Serving GPRS Support Node (SGSN). The SGSN establishes a tunnel through the *Gn* interface with a Gateway GPRS Support Node (GGSN) that provides connectivity to external networks. Through this path, the request message of a smart device enters the IP network

TABLE I
SUMMARY OF DATASET

Factor	Value
File Size	339.2GB
Number of Unique Devices	49,103
Number of Hosts	26,059
Number of HTTP Requests	2,025,994

and reaches the target server. Response data from the server to the device traverse in the reversed path. Therefore, we can capture all HTTP requests and responses with device identities and the accessing time between devices and web servers by high performance traffic monitor system (TMS) placed at the *Gn* interface.

In the above context of web browsing behavior and web traffic logs captured at the network-side, we define the main conceptual components used in the reminder of this paper:

Web traffic log. A web traffic log contains information about the web browsing actions and corresponding results of a group of devices. Such information includes the device identities, the requests submitted by the devices, the accessing time, and the responses of the requests. A typical web traffic log produced by the TMS device L is a set of records $l_i =< t_i, u_i, r_i, d_i >$, where t_i is the accessing time, u_i is an anonymized identity of a device that submitted the request, r_i is the submitted HTTP request, and d_i is the summary of the response like content length and content type. In our study, we do not use any information from the response contents of the requests. Thus, we denote a web traffic log with a number of records as $L = \{< t_i, u_i, r_i >\}$. Table I illustrates the summary of the dataset used in the paper for investigation and evaluation; the dataset was collected from the backbone of a cellular network in a Southern province of China on May 5, 2013.

Web objects and HTTP requests: In general, a web page p_i is composed by a collection of web objects, denoted as $p_i = \{o_j\}$. Each one of the web objects represents part of information of the page like an image or a multimedia file. A web object o_j can be identified and located by a unique identity named Universal Resource Locator (URL), which is the major part of the HTTP request r_j sent by the client to retrieve the object. Therefore, we can also denote the web traffic log as $L = \{< t_i, u_i, o_i >\}$.

Primary objects and primary requests: Commonly, a web page contains two types of web objects: primary object and secondary object. The primary object is the first object requested to retrieve the page by the devices, such as the objects requested by r_1, r_4 and r_6 in Fig. 1. Each web page has only one primary object. Other objects in the web page are the secondary objects, such as the objects requested by r_2, r_5 and r_7 in Fig. 1. The requests of retrieving the primary object and the secondary object are called primary request and secondary request, respectively. Usually, the primary request is triggered by devices accessing a URL via web API or opening a hyperlink in a web page. We call this kind of requests as *primary requests*, which are the key information source to reveal devices' behaviors. Other requests are defined as *secondary requests*.

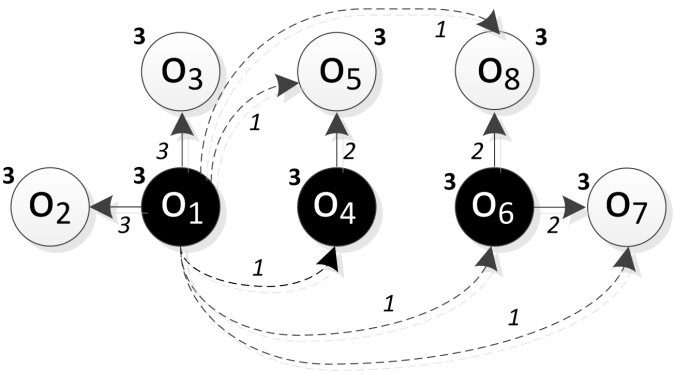


Fig. 3. Example of the request dependency graph

III. THE REQUEST DEPENDENCY GRAPH

A. Modeling Browsing Behavior with Request Dependency Graph

Based on the web browsing process and basic concepts described above, we introduce a dependency graph model to depict the dynamic web browsing behavior. Formally, we model the browsing behavior as a directed and weighted graph, $G = (O, S, E, W)$, referred to as a *Request Dependency Graph* (RDG). $O = \{o_1, o_2, ..., o_n\}$ is the set of nodes representing accessed objects which are identified by URLs. Each node o_i is assigned an occurrence count $S[o_i] \in S$ of the accessed object o_i. E is the set of directed edges with weights W. There is an edge from node o_i to o_j if and only if they meet the following conditions:

(i) For a request sequence $\upsilon = \{r_{i-1}, r_i, ..., r_j\}$ generated by the same device, in which accessed objects are $\{o_{i-1}, o_i, ..., o_j\}$, the interval between the accessing time of r_{i-1} and r_i is larger than τ, where τ is called the *lookahead time window*.

(ii) In the sub-sequence $\upsilon' = \{r_i, ..., r_j\}$ of υ, the interval between each pair of adjacent requests is smaller than τ.

A directed edge from o_i to o_j represents the dependency relationship between them. The weight of a directed edge is the number of times the pair $< o_i, o_j >$ appears in the measured HTTP request sequence R. A directed edge and the weight of the edge indicate the access of o_j followed by the access of o_i and the number of occurrences of such an action, respectively. Ideally, if all devices open web pages one by one with a given stay time and τ equals to the stay time, the edge from o_i to o_j indicates that o_j is an embedded object in the web page o_i. Unfortunately, this assumption is always violated by complicated browsing behaviors in the real world, such as the behavior of *Device* 3 in Fig. 1. Therefore, the request dependency graph derived from the web traffic log is more complicated than the simple structure of objects relationships defined by developers of the accessed web servers [8].

Algorithm I: Establish the Request Dependency Graph

Input:

$R = \{r_1, r_2, ..., r_n\} \rightarrow$ A set of HTTP requests

$r_i = \{u_i, t_i, o_i\} \rightarrow$ Data structure of each HTTP request

Output:

$G = \{O, S, E, W\} \rightarrow$ The dependency graph

```
1:  H = ∅ // The set of the request sequences of all devices
2:  for each r_i in R do
3:    push(H[u_i],r_i)
4:  end for
5:  for each h_i in H do
6:    o_predecessor = null
7:    t_last = 0
8:    for i=1 to length(h_i) do
9:      r_i = pop(h_i)
10:     t = getAccessingTime(r_i)
11:     o_i = getURL(r_i)
12:     O[o_i] = 1
13:     S[o_i]++
14:     if t − t_last > τ then
15:       o_predecessor = o_i
16:     else
17:       E[o_predecessor][o_i] = 1
18:       W[o_predecessor][o_i]++
19:     end if
20:     t_last = t
21:   end for
22: end for
```

Fig. 3 shows the request dependency graph derived from the browsing behavior example shown in Fig. 1. In the example, we have the same URL o_1 pointing to web page p_1 in request r_1, r_9, and r_{17}. Similarly, we get other nodes, from o_2 to o_8, and their occurrence numbers which are the bold numbers close to these nodes, such as 3 for o_1. By choosing an appropriate lookahead time window τ, which will be described in detail later, a request dependency graph can be built as shown in Fig. 3. Note that some edges start from o_1 and end at o_4 to o_8, as shown in dashed lines. These edges are caused by the parallel browsing behavior of *Device* 3, which results in short time intervals between all pairs of adjacent requests in the sequence from r_{17} to r_{24}. We will next introduce how we establish the request dependency graph to represent the dynamic behaviors and identify the primary requests from massive HTTP requests.

B. Establishing the Request Dependency Graph

The request dependency graph is initially empty and is established through a learning process, which is summarized in Algorithm I. The input data of the algorithm is a set of HTTP requests R. Each request r_i maintains information including the device identification u_i, the accessing time t_i, and the URL of the accessed object o_i. They are sorted in ascending order of the accessing time. The output of the algorithm is the request dependency graph G, which has a set of nodes O with occurrence counts S and a set of edges with weights W.

At first, the algorithm initializes an empty set H to save the set of the request sequences of all users at line 1. From line 2 to line 4, all requests are divided into a set of sequences. Each sequence is made up of requests from the same device, and ordered by the accessing time. Then, the sequence of each device h_i is processed one by one from line 5 to line 22. The accessing time and URL identity of each request are retrieved for further processing (line 10 and 11). The indicator of whether a node is in the graph is set to 1 (line 12), and the occurrence number of the node is incremented by one (line 13). As mentioned in Section II, each request may act as a primary request that is the first object requested to retrieve the web page or a secondary request of accessing the embedded object of a page. A secondary request is the successor of a primary request on the temporal dimension. So, the major part of the algorithm is to identify the predecessor and successor relationships between the HTTP requests. The codes from line 14 to line 19 are designed for this purpose. The time interval between a request o and the last request o_{last} $(t - t_{last})$ is compared with the lookahead time window τ (line 14). If it is larger than τ, the request o will be regarded as a predecessor request (line 15). Otherwise, the request will be regarded as a successor request. For each successor request, a directed edge is added from the current predecessor request to this request (line 17), and the weight of this edge is incremented by one (line 18). When a request is processed, the time for the last processed request will be updated (line 20). After having iteratively executed on request sequences of all users, the dependency graph is established.

TABLE II
METRICS OF THE STUDIED GRAPHS

Metric	$\tau = 1s$	$\tau = 3s$	$\tau = 5s$	$\tau = 7s$	$\tau = 9s$
$\|E\|$	166,217	217,236	239,121	251,103	258,741
$Density$	6.1E-6	8.0E-6	8.8E-6	9.2E-6	9.5E-6
$Centralization$	3.1E-3	4.8E-3	7.0E-3	1.1E-2	1.6E-2
$Heterogeneity^d$	5.2	6.7	7.8	8.8	9.9
$Heterogeneity^w$	20.6	19.3	19.1	19.4	20.1
$mean(d^{in})$	1.8	1.9	2.0	2.0	2.0
$\sigma(d^{in})$	27.6	47.0	59.1	66.4	70.0
$mean(d^{out})$	4.2	5.9	6.9	7.7	8.3
$\sigma(d^{out})$	102.2	321.3	573.3	864.0	1208.7
$mean(w^{in})$	2.8	3.1	3.3	3.4	3.4
$\sigma(w^{in})$	1784.6	2618.6	3075.6	3382.4	3585.1
$mean(w^{out})$	6.4	9.6	11.5	13.0	14.2
$\sigma(w^{out})$	4179.8	7560.0	10018.0	12555.7	15349.7
$\|C\|$	14,087	9,975	8,197	7,171	6,567

C. Characteristics of the Request Dependency Graph

We have established a number of request dependency graphs by applying Algorithm I on the dataset presented in Section II. For all the generated graphs, we have studied their characteristics from different aspects. Owing to the extra large size of all

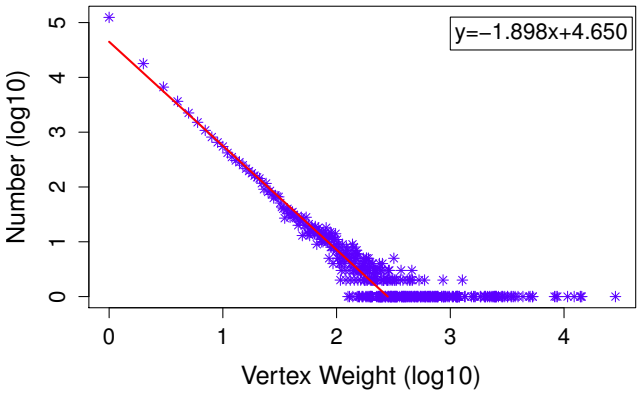


Fig. 4. Distribution of occurrence counts of nodes

the graphs and the results are similar at all time scales, here we present the analysis of a log piece of one hour from 9 AM to 10 AM on May 5, 2013. The total number of nodes, which are accessed web objects, in the studied graphs is 164,770. Table II presents the main characteristics of the different versions of the request dependency graph. Each version is built with a given value of the lookahead window τ in Algorithm I, from 1 second to 9 seconds. To comply with the evaluation method of the weighted directed graph, we normalize the weight of each edge as:

$$w'_{ij} = \frac{w_{ij}}{max(W)} \tag{1}$$

where $W = \{w_{ij}\}$ is the set of weights of each node. We can see that the number of edges increases when the lookahead window increases because more requests are treated as the secondary requests that connect to the predecessor primary requests when the lookahead window gets larger. However, it is noted that the graphs are sparse in all conditions, as indicated by the small values of densities in Table II. The density of the request dependency graph is defined as:

$$Density = \frac{2|E|}{|O| \times (|O| - 1)} \tag{2}$$

where O is the set of nodes and E is the set of directed edges in the dependency graph. A density close to 1 indicates that all the nodes are strongly connected with each other while a density close to 0 means an overall weak connection among nodes. We use the graph centralization to study how evenly the centrality distribution is among nodes in the request dependency graph. The graph centralization is defined as:

$$Centralization = \frac{|O|}{|O| - 2} \times \left(\frac{max(K)}{|O| - 1} - \frac{mean(K)}{|O| - 1}\right) \tag{3}$$

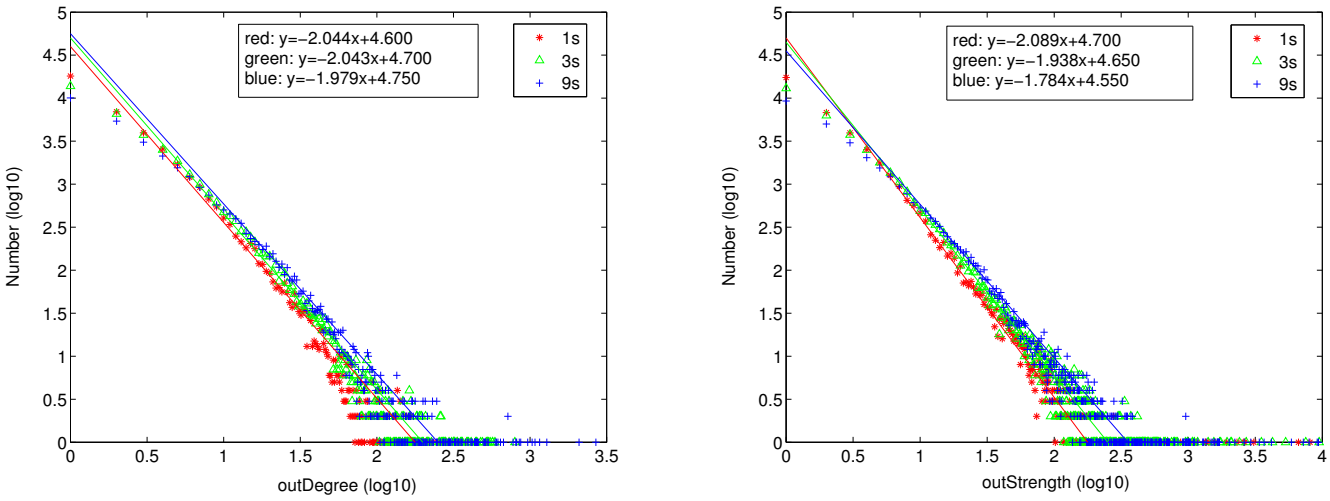
where $K = \{k_i\}$ is the set of the total number of degrees of each node and $k_i = \sum e_{ij}$. The small centralizations of all the graphs indicate that the topology of the request dependency graph is more like a grid than a star. The heterogeneity metric measures the variation of connectivity across the nodes in the graph. It is defined as:

$$Heterogeneity = \frac{\sqrt{var(K)}}{mean(K)} \tag{4}$$

where $K = \{k_i\}$ is the set of the total number of degrees or weighted degrees of each node and $k_i = \sum e_{ij}$ or $k_i = \sum w'_{ij}$. We calculate the heterogeneities of degree and weighted degree. The big values of heterogeneities imply that these graphs are highly heterogeneous. The heterogeneous characteristics of the request dependency graph are also indicated by the large variances of in-degree (d^{in}), out-degree (d^{out}), weighted in-degree (w^{in}) and weighted out-degree (w^{out}) as compared to their average values.

To study the heterogeneous characteristics of the request dependency graph in detail, we plot the distribution curve of several important metrics of the graph. All of the distributions of the node occurrence, which is the number of times of the web objects accessed, have extremely long tails spanning several orders of magnitude. The mean node occurrence is 4, but the maximum occurrence is 13,260 and the standard deviation of the distribution is approximately 78.1. That is, the distribution is so skewed that we are able to approximate the distribution with a power-law function $f(x) = \alpha \cdot x^k$. Fig. 4 shows the distribution of node occurrence and the fitted line using the power law, whose parameters are shown in Table III. Besides this kind of the skewed distribution of the web object accesses, which has been shown in previous works [9][10], we further investigate the relationships among the accessed web objects. In the definition of the request dependency graph, the out-degree and weighted out-degree of each node represents the number of the unique web objects followed by this node and the total occurrence of such relationship appeared in the traffic log, respectively. Fig. 5 shows the distributions of the out-degree and the weighted out-degree in the example request dependency graph. It is interesting to see that both of them exhibit a power law behavior. In addition, the curves of the distributions of the in-degree and the weighted in-degree, which depict how many individual predecessor nodes of a web object and the total number of such incidents, respectively, show the similar shape of those of the out-degree and weighted out-degree. The parameters of the fitted power laws of these four metrics are shown in Table III.

From the generated graphs, we note that the request dependency graph is a sparse graph with a strong clustering effect, which is shown in Fig. 6(a). It implies that the component structure is an interesting characteristic of the request dependency graph. A component of a graph is a subgraph in which any two nodes are connected by a path, and those nodes are connected to no other nodes in the super graph. Fig. 6(b) and Fig. 6(c) show two example components of the studied graph. The central node of the star shown in Fig. 6(b) is an entry point of a specific statistical function of doubleclick.net, the leading web access statistical service



(a) Out-degree (b) Weighted out-degree

Fig. 5. Distributions of the out-degree and the weighted out-degree in the example request dependency graph

provider. The nodes around the central node is a popular forum website in China, whose pages have the same embedded link pointed to the specific statistical function. Fig. 6(c) shows a component with a complicated structure, which contains a number of web objects of the web site 51tv.com, a popular online video website in China. The complicated topology of this component is caused by the complex relationships among the accessed objects in Fig. 6(c), which are homepage, login page, video list pages and other embedded web objects. To demonstrate how the request dependency graph can be applied to analyze the network traffic of WoT elements, we plot a cluster of web objects accessed by the 'Gaode' application, the leading mobile navigation application in China, in Fig. 6(d). The URLs of the center nodes are the RESTful APIs of the 'Gaode' web servers, such as webapi.ampa.com and restapi.ampa.com. The mobile clients use these APIs to report the positions to servers, and then retrieve map data and traffic status by accessing subsequent URLs, which are parsed from the response contents of these APIs and are the nodes around the center nodes.

To study the overall component characteristics of the graph, we transform the directed request dependency graph into an undirected graph by setting $e'_{ij} = e_{ij} \vee e_{ji}$. The plot in Fig. 7 shows the connected components distribution. Each point represents the number of components of a given size, which is expressed as the total number of nodes in the component. Also in this case, for all three values of τ, the distribution follows a power law, whose parameters are shown in Table III.

TABLE III
PARAMETERS OF THE FITTED POWER LAWS ($\tau = 3s$)

Power Laws: $f(x) = \alpha \cdot x^k$	α	k
Occurrence Count	4.65	-1.90
In-degree	4.70	-2.33
Out-degree	4.70	-2.04
Weighted in-degree	4.70	-2.11
Weighted out-degree	4.65	-1.94
Nodes of components	4.50	-2.65

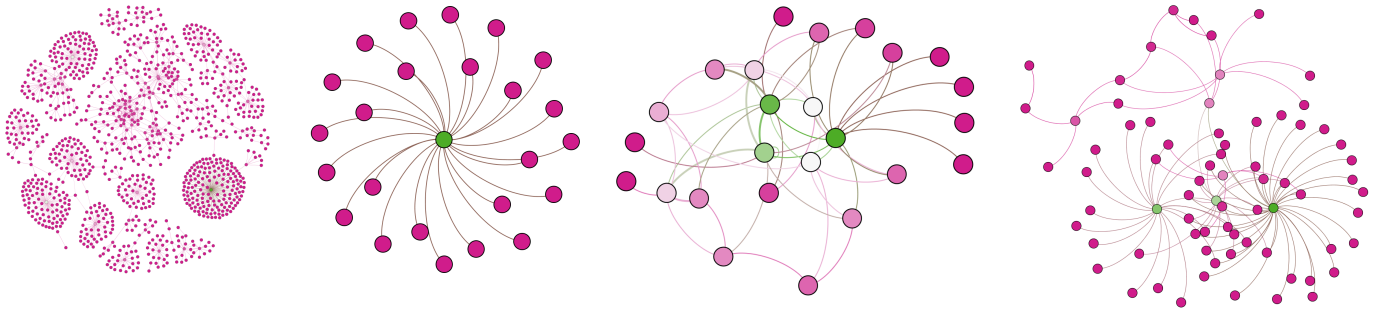
IV. PRIMARY HTTP REQUESTS IDENTIFICATION

We have utilized the request dependency graph to perform several tasks of the web usage mining like mining of the access patterns and website decomposition, and produced good results. In this paper, we take a fundamental web traffic preprocessing task, primary HTTP requests identification, as an example to demonstrate the effectiveness and potential of the proposed request dependency graph for web traffic data mining.

A. Problem Statement of Primary Requests Identification

A number of studies have shown that the set of initial HTTP requests of web pages is only a small part of the total web traffic [3][6]. Therefore, the quality of the web traffic mining significantly depends on the outcomes of an important preprocessing task on web traffic records, primary requests identification. Primary requests identification is a process of obtaining the set of requests triggered by users or devices from a large number of captured HTTP requests. It is important for various web applications, such as web search optimization [11][12], web usage mining [13][14], and anomaly detection [15][16].

As the web servers move from relatively static pages to functional carriers with embedded multimedia and dynamic contents, the nature of the interaction between devices and web servers changes as well. This results in great challenges to accurately identify primary requests in the following aspects. (1) Web pages have become increasingly complex because the number of embedded objects has increased [6]. Many requests occur not only as a result of the page click, but also as a



(a) Cluster effect (b) doubleclick.net (c) 51tv.com (d) amap.com

Fig. 6. Cluster effect of the request dependency graph and component examples

result of interactions between devices and web servers after the initial request. (2) The traditional browsing paradigm of visiting a sequence of web pages one by one in the same browser window has been greatly changed. Parallel browsing behavior is prevalent because almost all browsers are equipped with the multi-tab function to support opening more than one page from a single page [17]. (3) Advanced web technologies, such as AJAX, Flash, *etc.*, enable web server developers to put more functional scripts into the web pages. Dynamic requests triggered by these scripts are hard to be distinguished from opening web page action, and break the traffic patterns of original embedded objects in the page.

In Section II, we have shown some typical structures of modern web pages and the HTTP request sequences of accessing them in Fig. 1. To access a web page, a device usually generates a HTTP request with the URL of the web page. We call the initial HTTP request as the primary request. After receiving the response from the web server, the web client parses a set of URLs that point to the embedded objects in the page, named secondary objects. As a consequence, the web client generates a cascade of separated requests to download each of the secondary objects. Although we can describe such a simple process based on human perception, the primary requests cannot be determined straightforwardly at the network-side, in which only the HTTP request sequence $(r_1, r_2, r_3, ...)$ along with their own accessing time $(t_1, t_2, t_3, ...)$ is observed. To the best of our knowledge, few methods have been proposed in the literature to identify the primary requests, and are inadequate for today's complicated web traffic. Therefore, primary requests identification is an open issue and even becomes more difficult nowadays. Towards this end, we propose a novel primary requests identification algorithm based on the request dependency graph built from HTTP request logs. In this context, the primary requests identification problem is formulated as follows:

Primary Requests Identification: Given a request dependency graph $G = \{O, S, E, W\}$, which is established from a web traffic log $L = \{< t_i, u_i, o_i >\}$, the primary requests identification problem is to identify the set of objects $O_{primary} \subseteq O$ such that each object in $O_{primary}$ is accessed by a request corresponding to opening a web page.

B. Primary Requets Identification Algorithm based on the Request Dependency Graph

In the dependency graph, a node o_i represents the accessed object and the occurrence count $S[o_i]$ is the number of times this object has been accessed by devices. A directed edge from o_i to o_j represents that o_j is accessed right after o_i has been accessed. Obviously, we cannot conclude whether a request is a primary request only by the existence of an edge from other nodes to it. For example, in Fig. 3, o_4 cannot be identified as a secondary request by the existence of an edge from o_1 to o_4, which is caused by the parallel browsing behavior of *User* 3. We have to use other information, the weights of each edge to a node from other nodes. The sum of these weights represents how many times an object was accessed following other objects. Therefore, whether a request is a primary can be inferred by the probability p of a request o_i being the primary request, which is expressed as follows:

$$p = 1 - \frac{d^{in}_{o_i}}{S[o_i]} = 1 - \frac{\sum_{o_j \in O} W[o_j][o_i]}{S[o_i]} \tag{5}$$

The basic idea of inferring whether a request is a primary request is by thresholding, i.e., comparing p with a threshold ρ. If p is larger than ρ, implying that the request following other requests seldom occurs, the request can be identified as a primary request. Otherwise, it tends to be the request of an embedded object. Obviously, the number of requests identified as the primary requests depends on the value of ρ. A larger ρ results in fewer requests to be identified as the primary requests, but they are more likely to be true, i.e., the primary requests. To get a reasonable trade-off between the quantity and quality of identification results, we develop a self learning process to determine the value of ρ, which is summarized in Algorithm II. An initial threshold ρ_0 is set as a number close to 1, for example, 0.9. Nodes identified as the primary requests by this large threshold are most likely true. We put these primary nodes into a set $O_{primary}$. If a node has a directed edge from a node in $O_{primary}$ to it, this node is placed into another set $O_{secondary}$. After this, we can

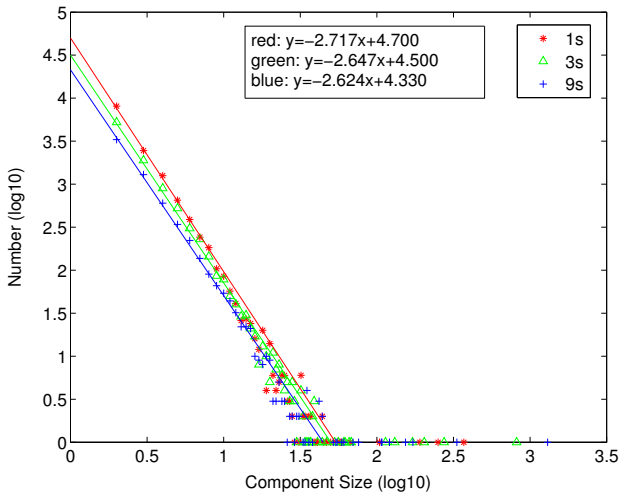


Fig. 7. Distributions of component sizes

calculate the expected ratio φ_0 of the primary requests in all requests, i.e., the number of actual primary requests over the total number of requests. Then, an iterative process is executed by varying threshold ρ from 1 to 0 with a step size of 0.01. For each ρ, we can obtain an identification result $O_{primary}$ and the corresponding φ. Obviously, we have $0 \leqslant \varphi \leqslant 1$ for $1 \geqslant \rho \geqslant 0$. The iterative process is accomplished when $\varphi \geqslant \varphi_0$. Then, we identify the current $O_{primary}$ as the set of primary requests C.

C. Experimental Evaluation of Identification Results

To evaluate the effectiveness of the proposed method, we collect a ground truth dataset for an experiment similar to [6][18]. We manually visit the top 5 popular websites of seven categories according to Alexa ranking [19]. For each of these websites, we randomly visit at least 50 links in them for up to 30 times by the Google Chrome browser. All HTTP requests generated by the browser are captured by the Wireshark tool as the ground truth dataset. The accessed URLs are exported from the history records of Chrome and used to mark the URLs in the dataset. By this way, we obtain an experimental ground truth dataset with 1,913 primary requests and 153,000 secondary requests. We use the *F1* score [20], which combines precision and recall factors with an equal weight, to measure the accuracy of identification results. The *F1* score can be calculated using Equation 6, in which the precision, $P = TP/(TP + FP)$, is the number of correctly identified primary requests divided by the number of all identified primary requests. The recall, $R = TP/(TP+FN)$, is the number of correctly identified primary requests divided by the number of requests that should be identified as primary requests. The F1 score reaches its best value at 1 and worst value at 0. We implement the primary identification algorithm as a program by leveraging the Hadoop MapReduce parallel computing model [21] to handle the massive web traffic data. Table IV shows the evaluation results. We can see that the F1

Algorithm II: Identify User Clicks

Input:
$G = \{O, S, E, W\}$ → The dependency graph
Output:
$C = \{c_1, c_2, c_3, ...\}$ → The set of primary requests

```
1: s_primary = 0
2: s_secondary = 0
3: for each o_i in O do
4:   d^in = 0
5:   for each o_j in O do
6:     d^in += W[o_j][o_i]
7:   end for
8:   p = 1 − d^in / S[o_i]
9:   if p ⩾ ρ_o then
10:    push(O_primary,o_i)
11:    s_primary += S[o_i]
12:  else
13:    push(O_secondary,o_i)
14:    s_secondary += S[o_i]
15:  end if
16: end for
17: φ_0 = s_primary / (s_primary+s_secondary)
18: for ρ=1 to 0 step 0.01 do
19:   s_primary = 0
20:   s_secondary = 0
21:   for each o_i in O do
22:     d^in = 0
23:     for each o_j in O do
24:       d^in += W[o_j][o_i]
25:     end for
26:     p = 1 − d^in / S[o_i]
27:     if p ⩾ ρ then
28:       push(O_primary,o_i)
29:       s_primary += S[o_i]
30:     else
31:       push(O_secondary,o_i)
32:       s_secondary++
33:     end if
34:   end for
35:   if (φ=s_primary/(s_primary+s_secondary)) ⩾ φ_0 then
36:     break;
37:   end if
38: end for
39: C = O_primary
```

scores of all seven categories are around 0.9; this proves that the proposed method is promising.

$$F1 = \frac{2 \cdot P \cdot R}{P + R} \tag{6}$$

In order to illustrate the better performance of the proposed method, we compare the identification accuracy of our method with the data cleaning method, which is widely used by existing commercial web traffic preprocessing systems. We execute our proposed method and the data cleaning method

TABLE IV
PERFORMANCE ON GROUND TRUTH DATASETS

Category	Kids	Society	Sports	Shopping	Business	Science	Arts
TP	218	198	241	281	170	218	236
FP	33	15	23	11	19	18	13
FN	37	49	31	119	9	27	80
P	0.868	0.929	0.912	0.962	0.899	0.923	0.947
R	0.854	0.801	0.886	0.702	0.949	0.889	0.746
F1	0.861	0.860	0.899	0.812	0.923	0.906	0.835

on the dataset captured from the real-world network, which is described in Section II. The identified user clicks produced by the two methods are compared with the real benchmark data derived by the human perception to verify the correctness of the results. For each method, we evaluate the identified clicks of 100 random users on three well-known news portal websites having rich embedded objects in pages, namely, sina.com.cn, sohu.com, and ifeng.com. Identified primary requests in six sample time slots, 9-10, 11-12, 13-14, 15-16, 17-18, and 19-20, are verified. F1 scores of the experimental results are shown in Table V, in which DC represents the Data Cleaning method and DG represents our Dependency Graph model based method. For the above three websites, the average F1 scores of our method are 0.919, 0.899 and 0.818, respectively, (rather close to the best value of their respective F1 scores), while those of the data cleaning method are only 0.322, 0.282 and 0.343, respectively. The results clearly demonstrate that our method is more accurate than the data cleaning method.

In Algorithm I, the lookahead time window τ is used to determine whether a request is an initial primary request for establishing the request dependency graph. In our method, whether a request is a primary request is inferred by comparing the statistical probability value p of being the primary request with a threshold ρ. To evaluate the impact of the value of τ on the accuracy of identification results, we conduct a set of experiments to study the *F1* score versus various values of τ. The evaluation result is shown in Table VI. We can see the *F1* scores of the above three websites remain relatively stable with various values of τ. This demonstrates that the accuracy of our method is not sensitive to the parameter τ. In the experiment, we choose $\tau = 3s$.

In order to demonstrate the self learning process of selecting the optimal threshold ρ in Algorithm II, we plot the values of

TABLE V
F1 SCORES OF IDENTIFICATION RESULTS

Website	Method	9-10	11-12	13-14	15-16	17-18	19-20	Avg
sina	DC	0.325	0.364	0.310	0.369	0.315	0.250	0.322
	DG	0.936	0.906	0.966	0.917	0.854	0.932	0.919
sohu	DC	0.283	0.267	0.331	0.328	0.187	0.292	0.282
	DG	0.958	0.894	0.949	0.789	0.901	0.901	0.899
ifeng	DC	0.343	0.369	0.382	0.288	0.345	0.331	0.343
	DG	0.850	0.732	0.876	0.790	0.772	0.889	0.818

TABLE VI
F1 SCORES OF VARIOUS LOOKAHEAD TIME WINDOW

Website	$\tau = 1s$	$\tau = 2s$	$\tau = 3s$	$\tau = 4s$	$\tau = 5s$
sina	0.898	0.931	0.936	0.928	0.920
sohu	0.958	0.950	0.958	0.924	0.924
ifeng	0.846	0.844	0.850	0.781	0.757

φ versus the threshold ρ as shown in Fig. 8. In the experiment, we set the initial threshold ρ_0 as 0.9. The expected ratio φ_0 of primary requests in all the requests corresponding to ρ_0 is 0.0337 (the red dotted line in Fig. 8). The ratio φ gets larger with decreasing value of ρ, and reaches 0.0360 when ρ is 0.72, which is the optimal threshold for our experiment. In Table V, values of the *F1* score indicate that $\rho = 0.72$ leads to a reasonable trade-off between precision and recall in results. The results demonstrate that the proposed identification method can achieve high precision and do not depend on manually choosing the right parameter values of the algorithm.

V. RELATED WORKS

Web traffic logs are widely considered as a valuable resource to understand web clients' behaviors and interests. The main challenges in analyzing web traffic logs lie in two aspects: extracting the relationships among HTTP requests from the raw logs, and identifying the set of HTTP requests that can represent web clients' behavior. To our best knowledge, no one has addressed the specific problem tackled in this paper, but our work is related to previous works in the areas of the above two aspects, specifically, the graph model of web traffic and the primary requests identification. We describe these related works as below.

Graph model of web traffic. As the primary way to understand the navigation behaviors and preferences of the web users, web usage mining has attracted ample attention in recent years. The most important task in web usage mining is to build an effective data model to represent the relationships among the accessed web objects. One main research focus attempts to infer the hidden relationships among web objects by projecting the raw lists of HTTP requests over different types of graphs, a natural way to represent relations among observed objects. For example, we modeled the affinity relationships among websites as a graph to identify website communities [22]. After abstracting the web traffic logs as a graph, a set of graph theory based mathematical tools can be applied to study the graphs for different purposes, such as web content prefetching and frequent usage pattern mining.

Predictive web prefetching is an effective technique to reduce the latency of retrieving web contents by prefetching web objects before the user requests them. Besides the prefetching algorithm, the key point of web prefetching is building a data model to represent the historical patterns of accesses to the web objects. Considering that structures of websites and navigation behaviors of users are commonly determined by the hyperlinks in web pages, a graph model is a natural way to view relations among accessed web objects. The dependency

graph (DG) model was firstly proposed to depict the access pattern of web objects for web prefetching by Padmanabhan and Mogul [23], which was inspired by a prior work of predicting file accesses [24]. A node in the dependency graph is an web object that has ever been accessed. There is a directed edge from a node A to another node B if and only if B was accessed within w accesses after A. The weight of the edge from A to B is the ratio of the number of accesses to B within w accesses after A to the number of accesses to A itself. Based on the graph, the prefetching controller can prefetch the web object B if A is accessed and the weight of the edge from A to B is higher than a pre-defined threshold, which can be dynamically set to different values in various environments. However, the noticeable increase of the number of embedded objects per page in the modern web environment has reduced the effectiveness of the simple DG based algorithm. To overcome this problem, Domenech *et al.* [8] proposed an enhanced graph model named Double Dependency Graph (DDG), which differentiated two classes of dependences between objects of the same page and objects of different pages. The DDG based algorithm can improve prefetching performance by reducing useless predictions of embedded objects. Because the goal of both DG and DDG is to illustrate the access patterns of web objects, they only considered the steps between accesses, and ignored the time lapse between accesses, which is an important factor to infer the relationships among HTTP requests. Referrer Graph (RG) was another graph model proposed to depict access patterns of web objects [25][26]. It was built from access logs by using the object URI and referee field in each HTTP request. Although RG is a more efficient solution as compared to DG and DDG by reducing the number of edges in the graph to save computational and memory resources, it relies on the referrer field of the HTTP request, which is not available in the data captured at the network-side owing to the dramatically increased complexity and workload on the packets inspection devices.

Primary requests identification. Web traffic has been proved to be an invaluable asset for Internet Service Providers (ISPs) to enhance user experience and increase revenue by web usage mining [27]. Compared to business owners of websites, ISPs have a unique advantage that they can observe all traffic passed through their network elements. It enables them to expose web clients' behavior and interests by analyzing web traffic. The web traffic logs consist of two types requests, the primary requests that are the initial requests to open web pages, and the secondary requests that are requests automatically generated by web clients to retrieve embedded objects. Obviously, the primary requests are more important than the secondary requests to understand which contents are interested to users. However, as compared to the large fraction of the secondary requests, the primary requests are a small subset of web traffic logs [6]. Therefore, primary requests identification is a critical technical issue to be addressed before performing web usage mining.

In recent years, several methods have been proposed to identify the primary requests, and they can be categorized into four types. The first type is the time-based approach [28]. It assumes that each secondary request always appears after a primary request in a short time. As such, a request after a given inactive time period can be considered as a primary request. Although the time-based approach is a straightforward method, the estimation of the time threshold is not an easy task because of complex network conditions. The second type is the data cleaning based approach, which filters requests by the type of accessing files and removes requests for unwanted file types such as images and multimedia files, which are considered as the secondary requests [29][30]. Although it is simple to implement this kind of methods, they can hardly achieve accurate identification and suffer greatly from the increasing complexity of file types. The third type is to identify the primary requests by means of clustering techniques [31][32]. These methods represent requests as vectors by their attributes or accessing time, and apply clustering techniques to identify the primary requests. Owing to the limitation of the sensitive parameter (the number of clusters k) and the complex iterative computational model, the clustering based methods are not applicable in the situation with massive request records. The fourth type is the model-based approach [16][33][34], which abstracts web browsing behavior as a mathematical model, such as dynamic Bayesian networks and hidden semi-Markov models, and uses a set of training data to obtain the key parameters of the model. Then, the trained model is used to identify the primary requests. These methods usually make some assumptions to establish the models; for example, a primary request is only determined by the previous primary request. However, the prevalent parallel browsing behavior by using the multi-tab browsers breaks these assumptions and renders these methods ineffective. To improve the performance, some researchers proposed to combine a set of filters like Referer-based, time-based and content-type-based filters to perform the primary requests identification task [6][18]. Although the precision of identification can be improved for some cases, the proposed combination approach cannot overcome the defects

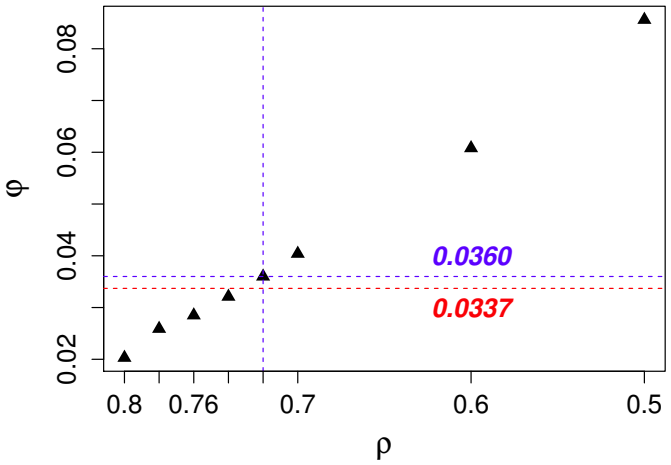


Fig. 8. φ values of various ρ

of the underlying filters.

VI. Conclusion

In this paper, we have proposed a request dependency graph to model the complicated web browsing behavior in the WoT environment. We have developed a methodology to establish the request dependency graph by processing the sequence of HTTP requests. We have presented an extensive analysis of a large graph derived from the traffic log containing millions of requests. Several interesting characteristics of the request dependency graph have emerged from the analysis. In particular, the graph appears to be weakly connected, decentralized, heterogeneous, and a number of its measures are governed by power laws. Then, we have shown a key application, primary requests identification, in the web usage mining that can be effectively tackled by the request dependency graph. We have developed a primary requests identification algorithm from massive HTTP requests by a self learning process based on the graph model. We have conducted an experiment to evaluate the proposed method based on the dataset collected from a real world cellular IoT network. Experimental results have substantiated that our method achieves higher accuracy as compared with the widely used data cleaning method. Browsing behavior modeling and primary requests identification are fundamentally critical for subsequent web usage mining. We expect our work will enhance the quality of web usage mining, and benefit the analysis of user behaviors and interests to improve network and web server performance. In addition, several works will need further research: (i) finding a way to decompose and visualize the large and complex request dependency graph built from massive traffic logs; (ii) studying the distribution of embedded objects in web pages of modern WoT environment based on the identification results; (iii) exploring more applications based on the request dependency graph.

References

[1] K. Ashton, "That 'internet of things' thing," RFiD Journal, vol. 22, no. 7, pp. 97-114, 2009.

[2] D. Guinard, "A Web of things application architecture," Dissertation, Eidgenossische Technische Hochschule ETH Zurich, Nr. 19891, 2011.

[3] F. Schneider, B. Ager, G. Maier, A. Feldmann, and S. Uhlig, "Pitfalls in HTTP traffic measurements and analysis," In Passive and Active Measurement, Springer Berlin Heidelberg, pp. 242-251, Jan. 2012.

[4] M. R. Meiss, F. Menczer, and A. Vespignani, "Structural analysis of behavioral networks from the Internet," Journal of Physics A: Mathematical and Theoretical, vol. 41, no. 22, 2008.

[5] P. Gill, M. Arlitt, N. Carlsson, A. Mahanti, and C. Williamson, "Characterizing organizational use of web-based services: Methodology, challenges, observations, and insights," ACM Transactions on the Web, vol. 5, no. 4, pp. 19, 2011.

[6] S. Lhm, V. S. Pai, "Towards understanding modern web traffic," In Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference, pp. 295-312, 2011.

[7] J. Liu, C. Fang, and N. Ansari, "Identifying user clicks based on dependency graph," 23rd IEEE Wireless and Optical Communication Conference pp. 1-5, May 2014.

[8] J. Domenech, J. A. Gil, J. Sahuquillo, and A. Pont, "DDG: An efficient prefetching algorithm for current web generation," 1st IEEE Workshop on Hot Topics in Web Systems and Technologies, pp. 1-12, Nov. 2006.

[9] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins, "Extracting large-scale knowledge bases from the web, " International conference on Very Large Data Bases, vol. 99, pp. 639-650, Sep. 1999.

[10] L. A. Adamic, B. A. Huberman, "The nature of markets in the World Wide Web," Quarterly Journal of Electronic Commerce, vol. 1, no. 1, pp. 5-12, 2000.

[11] U. Lee U, Z. Liu, and J. Cho, "Automatic identification of user goals in web search," In Proceedings of the 14th international conference on World Wide Web. ACM, pp. 391-400, 2005.

[12] Y. Zhang, W. Chen, D. Wang, and Q. Yang, "User-click modeling for understanding and predicting search-behavior," In Proceedings of the 17th ACM international conference on Knowledge discovery and data mining, pp. 1388-1396, 2011.

[13] O. Nasraoui, M. Soliman, E. Saka, A. Badia, and R. Germain, "A web usage mining framework for mining evolving user profiles in dynamic web sites," Knowledge and Data Engineering, IEEE Transactions on, 20(2), pp. 202-215, 2008.

[14] T. Pamutha, S. Chimphlee, C. Kimpan, and P. Sanguansat, "Data Preprocessing on Web Server Log Files for Mining Users Access Patterns," International Journal of Research and Reviews in Wireless Communications, vol. 2, 2012.

[15] G. Oikonomou, J. Mirkovic, "Modeling human behavior for defense against flash-crowd attacks," IEEE International Conference on Communications, pp. 1-6, 2009.

[16] Y. Xie, S. Z. Yu, "A large-scale hidden semi-Markov model for anomaly detection on user browsing behaviors," IEEE/ACM Transactions on Networking, vol. 17, no. 1, pp. 54-65, 2009.

[17] J. Huang, R. W. White, "Parallel browsing behavior on the web," In Proceedings of the 21st ACM conference on Hypertext and hypermedia. ACM, pp. 13-18, 2010.

[18] Z. B. Houidi, G. Scavo, S. Ghamri-Doudane, A. Finamore, S. Traverso, and M. Mellia, "Gold Mining in a River of Internet Content Traffic," In Traffic Monitoring and Analysis, Springer Berlin Heidelberg, pp. 91-103, 2014.

[19] Alexa - Actionable Analytics for the Web, http://www.alexa.com/ .

[20] C. J. Rijsbergen, Information retrieval, 1979.

[21] Apache Hadoop, https://hadoop.apache.org/.

[22] J. Liu, N. Ansari, "Identifying website communities in mobile internet based on affinity measurement," Computer Communications, 41, pp. 22-30, 2014.

[23] V. N. Padmanabhan, and J. C. Mogul, "Using predictive prefetching to improve world wide web latency," ACM SIGCOMM Computer Communication Review, 26(3), pp. 22-36, 1996.

[24] J. Griffioen, and R. Appleton, "Reducing File System Latency using a Predictive Approach," USENIX Summer, pp. 197-207, June 1994.

[25] B. de la Ossa, A. Pont, J. Sahuquillo, and J. A. Gil, "Referrer graph: A low-cost web prediction algorithm," In Proceedings of the 2010 ACM Symposium on Applied Computing, pp. 831-838, March 2010.

[26] P. Venketesh, and R. Venkatesan, "Graph based Prediction Model to Improve Web Prefetching," International Journal of Computer Applications, 36, 2011.

[27] J. Liu, F. Liu, N. Ansari, "Monitoring and analyzing big traffic data of a large-scale cellular network with Hadoop," IEEE Network, vol. 28, no. 4, pp. 32-39, July 2014.

[28] P. Barford, and M. Crovella, "Generating representative web workloads for network and server performance evaluation," In ACM SIGMETRICS Performance Evaluation Review, vol. 26, no. 1, pp. 151-160, June 1998.

[29] D. Tanasa D, B. Trousse, "Advanced data preprocessing for intersites web usage mining," IEEE Intelligent Systems, vol. 19, no. 2, pp. 59-65, 2004.

[30] K. R. Suneetha, R. Krishnamoorthi, "Identifying user behavior by analyzing web server access log file," International Journal of Computer Science and Network Security, vol. 9, no. 4, pp. 327-332, 2009.

[31] A. Bianco, G. Mardente, Mellia M, M. Munafo, and L. Muscariello, "Web user-session inference by means of clustering techniques," IEEE/ACM Transactions on Networking (TON), vol. 17, no. 2, pp. 405-416, 2009.

[32] Y. Fu, K. Sandhu, and M. Y. Shih, "A generalization-based approach to clustering of web usage sessions," Web Usage Analysis and User Profiling, Springer Berlin Heidelberg, pp. 21-38, 2000.

[33] O. Chapelle, Y. Zhang, "A dynamic bayesian network click model for web search ranking," In Proceedings of the 18th international conference on World wide web, pp. 1-10, 2009.

[34] C. Xu, C. Du, G. F. Zhao, and S. Yu, "A novel model for user clicks identification based on hidden semi-Markov," Journal of Network and Computer Applications, 2012.