Institutionen för datavetenskap
Department of Computer and Information Science

Final thesis

Identifying User Actions from Network Traffic

by

Georgios Rizothanasis

LIU-IDA/LITH-EX-A--15/036--SE

2015-06-08
Master’s Thesis

Identifying User Actions from Network Traffic

by

Georgios Rizothanasis

LIU-IDA/LITH-EX-A--15/036--SE

2015-06-08

Supervisor: Vengatanathan Krishnamoorthi
Department of Computer and Information Science
Linköping University

Examiner: Associate Professor Niklas Carlsson
Department of Computer and Information Science
Linköping University
Abstract

Identification of a user’s actions while browsing the Internet is mostly achieved by instrumentation of the user’s browser or by obtaining server logs. In both cases this requires installation of software on multiple clients and/or servers in order to obtain sufficient data. However, by using network traffic, access to user generated traffic from multiple clients to multiple servers is possible. In this project a proxy server is used for recording network traffic and a user-action identification algorithm is proposed. The proposed algorithm includes various policies of analyzing network traffic in order to identify user actions. This project also presents an evaluation framework for the proposed policies, based on which the tradeoff of the various policies is revealed. Proxy servers are widely deployed by numerous organizations and often used for web mining, so with the work of this project user action recognition can be a new tool when considering web traffic evaluation.
Table of Contents

1. Introduction ........................................................................................................................................... 7

2. Background and Related Work ............................................................................................................. 9
   2.1. User Action Identification .............................................................................................................. 9
   2.2. Behavior Models ............................................................................................................................ 9
   2.3. Web Page Performance Evaluation and Web Traffic Characterization .................................... 10
   2.4. Tabbed Browsing .......................................................................................................................... 11
   2.5. Browser Architecture .................................................................................................................... 11
   2.6. Web Page Loading ........................................................................................................................ 12

3. Methodology .......................................................................................................................................... 15
   3.1. Client Logs ................................................................................................................................... 15
   3.2. Proxy Logs .................................................................................................................................... 16
   3.3. Selenium-based User Models ....................................................................................................... 17
      3.3.1. Simple Browsing ....................................................................................................................... 17
      3.3.2. Tabbed Browsing .................................................................................................................... 18

4. Ground Truth Evaluation ...................................................................................................................... 21
   4.1. Firefox Extension Variations .......................................................................................................... 21
   4.2. Extensions Comparison .................................................................................................................. 22
   4.3. Ad/Tracking Related HTTP Traffic ............................................................................................... 24
   4.4. Browser Related HTTP Traffic .................................................................................................... 25

5. Proxy-log User Action Identification ................................................................................................... 27
   5.1. Time-based Policies ...................................................................................................................... 27
   5.2. HTTP-specific Policies .................................................................................................................. 30

6. Evaluation Results ................................................................................................................................ 33
   6.1. Number of Suggested User Actions ............................................................................................. 35
   6.2. Time-based Policies Evaluation .................................................................................................... 36
   6.3. Effect of Average Dwell Time ....................................................................................................... 38
   6.4. Tabbed Browsing .......................................................................................................................... 39

7. Discussion ............................................................................................................................................... 41

8. Conclusion and Future Work ................................................................................................................ 45

References .................................................................................................................................................. 47
1. Introduction

As Internet usage becomes more popular, the need to offer better network services becomes stronger. The end-to-end web performance depends on many factors that can be categorized in three categories: the server, the client, and the underlying network. A lot of research is done to improve web performance on all these three categories. One area of research that expands in all three categories is capturing and understanding the user behavior in order to be able to offer better services. In this direction behavior models are built and used to predict a user’s actions, thus helping the whole Internet infrastructure to be proactive in serving a user’s needs.

This project specializes in the subarea of browsing behavior. Browsing behavior models are the concept behind web caching, which is widely performed both on client level (with browser caching) and on the network level (with proxy caching). More specifically, the aim is to capture and understand a user’s browsing behavior from the network’s point of view: user actions are inferred through analysis of proxy logs which contain all the Hyper Text Transfer Protocol (HTTP) traffic that goes through it. In this direction, certain policies for recognizing and distinguishing user actions are proposed and their accuracy is assessed.

In order to accomplish this task, a working framework for user action detection and evaluation of the policies is implemented within this project. The main components of this framework are the proxy software needed for capturing the HTTP traffic and the client software needed for producing the ground truth against which the policies’ results are compared. A secondary component is a browsing model and its script implementation which serves as a generator of HTTP traffic. All these components and their complexity are presented within this project.

The proposed recognition policies are based on heuristics and rely on various information from the HTTP traffic such as timestamps and header field information. In order for the policies to be more adaptive, they are also dependent on tuning parameters. Finally, the proposed policies are tested under various scenarios of web browsing and the presented results take into consideration the tuning parameters of the different policies, as well as the effectiveness of applying groups of these policies.

User action identification is commonly performed either on the client or the server side. For the client side, browser instrumentation is required, which usually leads to a restricted number of clients to be used. For the server side, the deduced browsing model is specific to the server and the website it is serving. The contribution of this project is a proxy-based methodology that eliminates the need of a specific server or a specific set of clients in the process of understanding a user’s browsing behavior. Proxy logs that are easily accessible by a network administrator can thus be used for user behavior analysis.

Moreover, the results from identifying user actions can be used as an additional tool in network traffic studies. Being able to distinguish HTTP transactions that are a direct result of a user action from the ones that are invoked indirectly is valuable in plentiful contexts when trying to understand what services are actually requested by users rather than initiated by merely accessing a website. For example, even though it has been derived from proxy log analysis that a big portion of modern day web traffic is related to advertising, it would be interesting to investigate how much of this traffic is a result of user actions.

The remainder of this thesis is structured as follows. First, related work for user action identification and browsing behavior models, as well as background information about web browsing are presented in Chapter 2. All the necessary tools for this project’s research are described in Chapter 3, while their evaluation is done in Chapter 4. Chapter 5 introduces the proposed policies and Chapter 6 demonstrates their accuracy. Finally, Chapters 7 and 8 discuss and conclude, respectively, the presented results.
Chapter 1: Introduction
2. Background and Related Work

2.1. User Action Identification

Liu et al. [17] suggest a dependency graph model for identifying user actions from proxy logs. HTTP requests belonging to a specific user are given two weights. The first weight corresponds to how many times a particular request has appeared in the log. The second weight reflects how many times this particular HTTP request is a primary request. Primary requests are those which occur after a certain time threshold since the last request. The decision of whether a primary request is a user action is based on the relation of these two weights and a probability threshold.

Their work has similar goals as this project and the timing threshold is also used by this project’s proposed policies. However, their results are based on analysis of traces from specific websites. More specifically, they use data from sina.com.cn, sohu.com and ifeng.com which are all news portal websites. Furthermore, the identification of user actions can only happen after the traces were collected and not live as the users browse the web. Finally, they build the ground truth by manually analyzing server logs. This project addresses all these points and follows a more general approach, not focusing on a distinct website or a distinct category of websites. Additionally, a working framework for automatically building the ground truth allows for a broader set of user actions than merely link clicks.

Another way of tracking user actions is by using the referer field of the HTTP header. The referer field of an HTTP request shows the Uniform Resource Locator (URL) of the HTTP request that led to this HTTP request. For example, Huang and White [15] used the referer field to identify the sequence of web pages in their examination of tabbed browsing behavior. In particular, if a page on a browser has a URL that will be the referer field for later pages and if the later pages have a different tab ID, then a tabbed browsing event can be recorded. If the tab ID remains the same, then a direct link clicking event can be recorded. This method works because the tracking mechanism was installed in the browser itself, so there was access to the final loaded pages. However, it does not fit a proxy-based data collection.

In order to collect data for user action identification most of the research so far uses client- or server-based techniques. However, gathering data from a client requires special software to be installed on the user’s machine, usually in the form of a browser extension as well as the consent of the user themselves. This limits the size of the gathered data [2], [4], [5], [8], [13], and [15].

On the other hand, gathering information on the server’s level offers a bigger variety of users and data, but the analysis of the information can only result in a behavior model specific for the website that is served by the chosen server [1], [7], [9], and [14]. Accessing server logs from multiple organizations requires also the consent of these organizations, which is even more challenging to acquire than the consent of users.

Therefore, analyzing data from a proxy can offer both a great length of data from various users and a more general browsing behavior model. Moreover, the logging software has only to be installed once in the proxy server and any organization can use its own network traffic data for the sake of performing user action analysis.

Furthermore, in order to obtain a basis to create a behavior model, most of the time log data are used. This is the case with the current project as well. However, user studies have also been used. A user study is when the browsing behavior of specific users is monitored and the users are also interviewed regarding the motivation behind their actions [4], [24]. This provides a better understanding of the browsing behavior and helps the researches build better behavior models. Consequently this could lead in better software/hardware solutions as is the case with tabbed browsing in the Firefox web browser [4].

2.2. Behavior Models

Analyzing a user’s web browsing behavior has been researched from various aspects, including revisitation of web pages [5], navigation within a website [1], and interaction with search results [13]. Part
of the research is often proposing a model that fits the specific goal. Various models have been proposed, each focusing on different aspects of the browsing procedure.

In general, visiting web pages can be seen as a stochastic process with web pages as the states. More specifically, a user will spend some probabilistically distributed time on a web page and then will select the next page according to some probabilistic distribution. Some of the stochastic models that have been proposed include: first-order discrete-time Markov chains [19], first-order continuous-time Markov processes [3] and ergodic Markov chains [10]. The first-order Markov chains take into consideration only the current page a user is viewing in order to predict the next page. Since no information about the browsing history are considered, the first-order Markov chains are memoryless models. On the other hand, a kth-order Markov model would take into consideration the last k pages the user has visited. Combinations of the previous models both by themselves and with other models have also been a frequent proposal, as for example: selective kth-order Markov models [7], dynamic nested Markov models [9], as well as maximum entropy and Markov mixture models [1].

In the area of behavior models a lot of research focuses on web search and information retrieval. Carterette et al. [18] use logs from a search engine to build a distribution of parameters modelling a user that steps down search results. White and Drucker [13] investigate behavioral variance regarding search-related activities on the Web and suggest two extreme classes of users. The first class tends to visit websites within a certain domain and also revisit frequently the same domains. On the other hand, the second class tends to visit more and new domains. Finally, Choo et al. [24] view web browsing as an information retrieval task and present different behavior models corresponding on different goals of the information retrieval process.

2.3. Web Page Performance Evaluation and Web Traffic Characterization

The performance of a web page loading in a client has been under the scope of research for a long time. Important factors that affect the performance are the download duration and the perceived latency by the user. Most of the research focuses on the download duration. For example, Krishnamurthy et al. [16] studied the relationship between amount of content in a web page and the time needed for the web page to be fully downloaded. Furthermore, Gill et al. [20] examined among other things the number of successful and redirection messages, as well as the size of requested objects that were due to web page downloading. A high number of redirections and a big size of requested objects would mean a longer downloading period.

Butkiewicz et al. [22] studied many factors concerning a web page’s complexity. The number and type of objects within a web page, the popularity of a web page, the category a web page belongs to as well as the number of servers the browser needs to contact in order to download a page are some of the parameters that were taken into consideration. Moreover, there was a distinction between servers that belonged to the same provider as the page and the ones that did not. The impact of all these factors on loading time was next analyzed.

However, it could be that a user cares more about receiving fast the initial requested page and not the drag-along content. The initial requested page corresponds to the user action. Therefore, adding user action identification in the previous studies would have the benefit of obtaining separate results about download duration and perceived latency by the user.

In a wider context, the proposed policies could be also used in research regarding web traffic in general. For instance, in an attempt to characterize the usage of web-based services Gill et al. [20] presented various classifications of HTTP traffic regarding the type and the provider of this traffic among other. However, an interesting insight would be to show how these classes distribute to actual user actions or drag-along content. According to one of their results around 19% of the traffic is advertising related, but it could be useful to know how much of this traffic is actually initiated by users.
2.4. Tabbed Browsing

According to Huang and White [15] it is quite common to have multiple windows or tabs open during a browsing session. This way of browsing is referred to as parallel browsing in general, or tabbed browsing when only referring to the use of multiple tabs. In the research of Dubroy and Balakrishnan [4] about the Firefox browser most of the participants used multiple tabs at least twice as often as multiple windows. The Opera browser version 4 was the first browser to use multiple tabs within a single window back in 2000 and since then all major browsers have incorporated this capability [8]. On the contrary, strictly opening web pages one after the other in one tab in one window constitutes the linear browsing model.

Having more web pages open at once results in new possibilities for user actions. For example, a user can load two or more web pages simultaneously, as well as switch between opened pages. It also gives more possibilities to web authors. An external link could be forced to open in a new tab, leaving the current page still open in the user’s browser. Finally, the browser’s behavior can vary regarding parallel browsing. Some browsers can be configured to prevent loading a page in the background, while the focus is on another web page.

The plethora of possibilities renders the user action extraction out of a log into a difficult task. Viermetz et al. [6] propose the construction of a user action tree with nodes representing web pages and the root-to-leaf paths representing possible web page sequences due to parallel browsing. Nonetheless, their results focus on the number of possible paths due to parallel browsing in relation to the total number of visited web pages and they do not include any quantification of the time between two parallel browsing actions or of the probability of parallel browsing actions.

2.5. Browser Architecture

In order to understand how the HTTP traffic is generated by a user action and its time distribution, it would be useful to describe how web pages are loaded by the browser. For that reason, some knowledge about a browser’s architecture would also be useful. Figure 1 shows a high-level overview of a web browser’s architecture [11].

![Browser Architecture](image)

Fig. 1. Browser Architecture

In terms of this project, the most interesting parts are the rendering engine, the network component and the JavaScript Interpreter. The rendering engine is responsible for parsing the Hyper Text Markup Language (HTML) document and the Cascading Style Sheet (CSS) resources, together with the results from the JavaScript interpreter in order to create the final displayed document. The network component is handling the HTTP traffic to and from the appropriate web servers. This HTTP traffic contains the necessary resources. The function of the JavaScript Interpreter is obvious.

Although, the focus of this project is the Firefox browser and its rendering engine, Gecko, this high-level architecture is common in all major browsers [11].


2.6. Web Page Loading

The classic scenario of a user request has the following steps:
1) The user requests a document/web page. This is usually an HTML document, although modern browsers can also display other types of documents, like a Portable Document Format (PDF) document.
2) The browser sends an HTTP request to the server for that document.
3) After receiving and processing the request the server sends an HTTP response back to the client/browser.
4) The browser then constructs the Document Object Model (DOM) tree and it will request the style sheet, script and other resources that are referenced by the document.
5) After receiving the style sheet resources the browser will apply them to the DOM tree.
6) After receiving the script resources the browser will execute them.

While the construction of the DOM tree is handled by a single main thread in the rendering engine, external objects can be downloaded in parallel. So, style sheets, scripts and images can be downloaded simultaneously by the network component. For typical browsers nowadays even more than 6 connections per hostname can be active at the same time [23].

It is important to mention here that this is rather an iterative process. From the browser’s perspective, while the DOM tree is constructed in the rendering engine, new external resources may be requested through the network component. If the newly arrived external resources are scripts that are executed by the JavaScript Interpreter this may result in manipulation of the DOM tree by the rendering engine.

This is also noticeable from the user. After the initial request of a web page, a user may start seeing some content, while new elements are gradually added. This is done to reduce the perceived latency of the whole page loading. Otherwise the user would have to wait until all the resources for the requested document are loaded and all the scripts are executed, with possible need of requesting more resources.

From all the above, it can be concluded that after a user action an initial document request will be executed which will be followed by a number of requests for more resources, like style sheets and scripts. These requests for resources translate to HTTP traffic. Furthermore, after the processing of these new scripts it may happen that new objects are added to the original document. It is possible that these new objects will also require additional resources and thus generate HTTP traffic and so on. Figure 2 shows this process.

![Fig. 2. Page Loading](image)

Furthermore, modern web pages offer a rich content, which means loading of a lot of resources. In order to minimize the overall latency of loading the page and especially the perceived latency by the user, modern web page authors implement various techniques. Facebook for example uses BigPipe\(^1\). The inspiration of this method lies in the processor pipelining and it basically introduces two new concepts to

---

\(^1\) Facebook BigPipe: [https://www.facebook.com/notes/facebook-engineering/bigpipe-pipelining-web-pages-for-high-performance/389414033919](https://www.facebook.com/notes/facebook-engineering/bigpipe-pipelining-web-pages-for-high-performance/389414033919)
Chapter 2: Background and Related Work

The classic page request scenario. First, a web page is fragmented in smaller sections which are treated as separate pages with their own style sheets and scripts. Then, the stages of loading these fragments can be pipelined. For instance, a fragment can be displayed to the user, while the CSS of another fragment is being downloaded.

Even by implementing these modern techniques of page loading, the generated HTTP traffic follows the same pattern. After a user action, there is an initial document request, followed by some HTTP traffic for this initial document. Then, bursts of HTTP traffic are generated as new objects are gradually added to the initial document. Depending on the techniques used by the web author of the web page, these bursts can be more or less intense and depending on the content size of the web page these bursts can continue for a longer time or stop rather quickly.

To visualize the above, Figures 3 to 5 show the loading of three web pages: huffingtonpost.com, nytimes.com, and buzzfeed.com. These websites were selected because they create a big amount of requests and they have similar loading durations. The y axis represents the number of HTTP requests that have been sent by the browser, while the x axis represents the time elapsed from the initial request in seconds.

In order to get a better understanding about the nature of these requests a distinction was made. The lower curve in the plots represents HTTP requests that are addressed to second-level domain names with the most and second most objects for each site. The upper curve represents the total number of HTTP requests.

The bursts of HTTP requests can be seen preferably on the nytimes and buzzfeed websites. Their diagrams have a more step-like look. For the nytimes website there are more bursts which are spread in time, but that is partially because it consists of more objects. On the other hand, when examining the total
number of HTTP requests for the huffingtonpost website, a smoother graph can be observed. This means, that bursts happened more often, but they were less acute.

One more thing that can be noted from the diagrams is the impact of the most dominant domains. The dominant domains are the two second-level domain names that were the most frequent among the HTTP requests. For the three sites presented above, the dominant domains more or less match the domain name of the requested website. For example, for the huffingtonpost.com website, the dominant domains were huffingtonpost.com and huffpost.com. All the other domain names present in the HTTP requests cannot be directly matched to huffingtonpost.com just by looking on their name.

For all three sites, it is obvious that the loading of the page starts with fetching objects from the dominant domain. Furthermore, after a while it is mostly the non-dominant domains that contribute objects. This is particularly clear for the huffingtonpost website. This website has a lot of external resources and more than half of its objects are fetched from non-dominant domains. Table 1 summarizes the total number of requested objects (which is the same thing as the total number of HTTP requests) and the total number of different domains for each of the three websites.

The objects per domain index shows that in order to fetch the same amount of content, the first website depends on much more servers. The objects per domain index is only useful for comparison between websites and it does not give insight on how the objects are distributed among the domains for a specific website.

<table>
<thead>
<tr>
<th>website:</th>
<th>huffingtonpost.com</th>
<th>nytimes.com</th>
<th>buzzfeed.com</th>
</tr>
</thead>
<tbody>
<tr>
<td>objects:</td>
<td>408</td>
<td>232</td>
<td>145</td>
</tr>
<tr>
<td>domains:</td>
<td>113</td>
<td>39</td>
<td>25</td>
</tr>
<tr>
<td>objects per domain:</td>
<td>3.61</td>
<td>5.95</td>
<td>5.80</td>
</tr>
</tbody>
</table>

*Table 1. Objects per domain.*
3. Methodology

The high-level scenario under which this project was developed is a user browsing the web with a client machine that is connected to a proxy server before accessing the Internet. The methodology that was followed can be divided into four steps:

- create ground truth at the user,
- capture HTTP traffic on the proxy,
- analyze and group HTTP traffic according to proposed policies, and
- compare results of analysis against ground truth.

![High-level Methodology](image)

The term ground truth refers to a client log file. This log file contains all the HTTP traffic that was generated by the specific client and more importantly it keeps this traffic grouped according to user actions that actually happened in the client. Obtaining the ground truth is an important piece of this project. In order to assert whether the discovery of user actions from the proxy data according to a certain policy is successfully done and to which extent, these user actions will be compared against the user actions found in ground truth. Therefore, it is essential that the ground truth is an accurate representation of the user’s actions.

The methodology is centered on two main components. The first component is a Firefox extension that records all HTTP traffic made from a browser during a user session and groups it according to the user actions. This is used as the ground truth. The second is a Squid proxy that records all the traffic that goes through it and produces the proxy log. Apart from the main components, a navigation tool for browsing the web is also needed. These components are next described in detail.

3.1. Client Logs

The client logs that are used as ground truth are collected with the Firefox extension. The Firefox extension is responsible for recording all the HTTP traffic made from the browser. The HTTP traffic is because of a user browsing the web with Firefox. The user performs various actions which result in network activity. These actions can be the loading of a new webpage, clicking on a link which will result in a new page load, using the back button of the browser and many more. The recorded HTTP traffic is grouped according to the user actions, that is all the HTTP requests and responses have a tag that corresponds to a user action. For example, if a user requests the webpage http://www.a.com then all the HTTP traffic that was needed to display this webpage will be tagged with a unique id number. Later when the user clicks on a link that will result in the loading of the http://www.a.com/new_data webpage, then all the HTTP traffic needed to display this new page will be tagged with a new unique id number.

It should be clarified here that HTTP traffic can be caused by a webpage without any user intervention. This is the case when a web page is automatically reloaded as a whole or some part of it or when a server
Chapter 3: Methodology

pushes updates to the client. The HTTP traffic that comes from these webpage actions will not get a new tag. The tag from the initial page load will remain.

There are numerous ways to interact with modern web pages. Apart from clicking on links, a user can load new objects by just scrolling down to a web page, clicking on a specific area of the web page or using a combination of keystrokes. Moreover, modern browsers offer search services from within the address bar, so a user can load the results of a search engine without having to load the search engine’s page first. However, the focus of this project is on web navigation from one web page to another rather than interaction within a certain web page. Thus, the interesting user actions are typing a URL in the address bar and clicking on links from one web page to another. The Firefox extension was built based on these actions.

Firefox Extension Structure

The extension has four main components. First is the HTTP observer. This component captures all the HTTP traffic that comes/goes from/to the Firefox browser. This component also retrieves various information from the HTTP traffic and is also responsible for matching requests and responses.

Second is the event listener component. This component waits for user actions and helps in the recognition of the initial HTTP request that results from that action. It consists of multiple event listeners. Event listeners are a very common way to identify interaction with a web page and there are a lot of them, but for this project the following ones were used:

- a location changed listener to identify a new page load,
- a click listener to identify clicking on a link,
- an address bar listener to identify the typing of a URL in the address bar, and
- a history access listener to identify going back or forward in the browser’s history.

The third component is the counter manager. This component is responsible for giving the correct counter to all the HTTP traffic. The counter serves as the unique ID tag for the user actions. In other words the counter manager performs the grouping of the HTTP traffic according to user actions. The counter manager receives information both from the HTTP observer and the event listener in order to decide in which action the HTTP traffic belongs to. It can be said that the counter manager is somewhat the brain of the extension where all the logic for the HTTP grouping happens.

The counter is per session. In the scope of this project, a browser session means the duration of opening a Firefox instance until all Firefox instances are closed. When all Firefox instances are closed the counter is reset. This is enough for this project because all the tests will be run as a single browser session. Moreover, this is an easy way to create a log file per session, since new sessions will start with a reset counter. The goal of the extension is to distinguish different user actions within a single browser session, so handling multiple browser sessions would not offer further benefits.

One of the core functions of the counter manager is the discovery of which tab the traffic comes from. This way two or more tabs can be loaded simultaneously and their HTTP traffic will be labeled as different actions. The same goes with two or more different browser windows that are open simultaneously. The counter is global and unique so actions from the first window will always have a different counter number than actions from the second window.

The fourth component is the output file manager. It includes all the necessary actions to open the output file and log all the HTTP traffic in the correct format.

3.2. Proxy Logs

The proxy logs are collected with the help of Squid. The Squid software is installed on a proxy server between the machines that their network activity should be recorded and the Internet. A Squid proxy is optimized for caching, but here the focus is on recording all the HTTP traffic that goes through it. The logging capabilities of Squid allow for recording of various information of the HTTP traffic. For example, the length of the HTTP response and the value of the referer field if it is available can be obtained. Squid
logs an HTTP request-response as a pair, but specific information about the request and the response can be extracted, for example from the HTTP header fields.

A particular interesting feature is the recording of time on the Squid proxy. This will be used later for the time based policies of distinguishing user actions. As mentioned before, Squid keeps one entry for an HTTP request-response transaction and it timestamps it at the end of the transaction. More specifically, the time the whole response has been sent back to the client is recorded. That is the response timestamp. However, Squid can also log the duration of a transaction and subtracting this duration from the response timestamp is how the timestamp of an HTTP request is calculated.

Additionally, Squid can also log the IP address of a machine that generates HTTP traffic. The IP address will be used to distinguish the different users. As a matter of fact, the first step in the proxy log analysis is to group the recorded HTTP traffic according to different users.

A special case of the HTTP traffic is HTTP Secure (HTTPS) traffic. HTTPS uses Secure Sockets Layer /Transport Layer Security (SSL/TLS) to encrypt all the content of the HTTP communication and protects against eavesdropping and tampering. Therefore, Squid cannot intercept HTTPS connections in order to obtain and log various information regarding them. In fact, traffic is tunneled between client and the appropriate server. The tunneled traffic passes through Squid, but Squid is only able to see that a Transmission Control Protocol (TCP) connection was made to a specific server, but is unable to see all the objects that were requested through this connection. A man-in-the-middle attack would allow for interception of the HTTPS connections in the Squid proxy, but that would break the whole concept of HTTPS. Instead, HTTPS traffic will be ignored for this project.

It should be noted here that Firefox should be configured to use a proxy server in order for the HTTP traffic to reach the Squid server. This is done by providing the IP address and the port number that Squid works with. In general this configurations should be done for all network traffic generated by Firefox, but for this project, as discussed before, the configuration is only needed for the HTTP traffic.

3.3. Selenium-based User Models

In order to create some client and proxy logs to test the proposed policies an automatic web navigation tool is employed. This tool is the Selenium web driver. With Selenium, browser automation is possible through a user defined script. The script can run for a user defined time which makes it suitable for creating extensive logs without any supervision. Since the implemented extension is for Firefox, the Firefox web driver from Selenium is used.

Furthermore, there is another reason behind using Selenium. As it was discussed before (section 3.1) not all interactions with a web page can be handled by the Firefox extension. Client logs from regular users would probably include user actions not supported by the Firefox extension. That would lower the accuracy of the ground truth. Nonetheless, restricting the Selenium script to only perform the supported actions eliminates this problem.

3.3.1. Simple Browsing

The automation script simulates a user's behavior while browsing the web and in order to do so a browsing model is needed. Modeling all the user's actions from the moment the browser is opened, during the web navigation and until the browser is closed is a complex procedure with a lot of details that need to be taken into account.

The model that fits this project is a first-order continuous-time Markov process. This is because it includes the interesting parameters while having a low complexity compared to the selective [7] or mixture models [1]. The interesting parameters are those that affect the network activity. Specifically, the time a user spends on a web page, also known as dwell time, affects the available time the browser has in order to load a page and the time distance between two consecutive user actions. Moreover, the Markov model assumes that web page accesses are memoryless, that is, each page access is independent of the

2 Squid HTTPS: http://wiki.squid-cache.org/Features/HTTPS
pages that were previously accessed. That fits well with this project because the proposed policies for action recognition are general and do not assume any relation among the web pages.

Another important parameter is the probability to open a new web page instead of clicking on a link because it affects the hosts involved in the network activity. Clicking on a link results in a higher probability that the next page will belong to the same domain as the previous page and/or part of the content will be shared by the two pages. Due to techniques such as persistent connections and content caching, a page that belongs to the same domain as the previous one will probably load faster. Other details like bookmarking a web page or deleting the browsing history are not important for this project since they do not create any network traffic.

According to the previous, the simple model of web browsing that was adapted for this project has the following steps.
1) The user requests an initial web page.
2) An exponentially distributed time is spent on this web page.
3) The users decides either to click on a link or request a new web page and the procedure continues with step (2).

![Fig. 7. Simple Browsing flowchart](image)

In Figure 7 a high level representation of the automation script is shown. Some implementation details were left out on purpose. More specifically, the ending condition is not present, so it looks like an infinite loop. For this project, the browsing procedure stops when a predefined number of loops has been reached.

The time a user spends on a particular website or dwell time is heavily dependent on the type of the website. In other words, having one exponential distribution for all websites is a far-fetched generalization. For example, Liu et al. [2] consider the dwell time as Weibull distributed and present all the parameters of the Weibull analysis in terms of ten different categories of websites.

However, since for this project the actual websites that are visited do not make any difference regarding the network traffic, provided they are not biased in any way, a single exponential distribution for the dwell time will be used regardless the web page under process. Moreover, the actual links that are followed by the users are also irrelevant since any of them is going to create some network activity anyway. Thus, the state transition probability would follow a uniform distribution.

Regarding the probability of requesting a new web page instead of clicking on a link, one of the most popular web navigation algorithms, the PageRank, proposes the damping factor [19]. According to PageRank when users browse the Internet they mostly follow the links from one webpage to the next one. However, with probability 0.15 they will stop this procedure and request a new web page instead.

### 3.3.2. Tabbed Browsing

In order to fit tabbed browsing, the previous model needs some modifications. In particular, additional interesting parameters are the probability of opening a link in a new tab as well as closing a tab, because
these parameters also affect the domains contacted for consecutive user actions and as discussed in the Simple Browsing section, this also affects the time between page loads.

Clicking on a link to open on a new tab while staying on the same initial page can be achieved in various ways. The following ways were tested and found to work for Internet Explorer 11, Mozilla Firefox version 24, Google Chrome version 41 and Opera version 28:

- Clicking on a link with the middle button of the mouse, which usually has also the scroll wheel on it. This is the reason why for this project clicking on a link to open in a new tab will often be referred to as middle-clicking.
- Holding the CTRL key and left-clicking on a link.
- Right-clicking on a link in order for the context menu to appear and then selecting the “Open link in new tab” option.

Another possible action when considering tabbed browsing is the possibility to switch tabs. However, this does not directly create network traffic and is basically the same as closing a tab and moving on to the next opened one. The only difference is that switching tabs means that both tabs remain open, but having multiple tabs open is already covered by creating a tab queue through middle-clicking links.

The new model has now the following steps.

1) The user requests an initial web page.
2) Some time is spent on this web page.
3) The user decides if the current tab will be closed. If the tab is closed then the user continues with the next tab in the tab queue and the procedure continues with the same step. Otherwise move to step (4).
4) The user decides whether to click on a link to open in a new tab. If this is the case then the new tab is added to the tab queue. It should be noted that the user does not leave the current tab and page. After some time is spent on the current page, the procedure continues with the same step. Otherwise, move to step (5).
5) The user decides either to click on a link or request a new web page and the procedure continues with step (2).

*Fig. 8. Tabbed Browsing flowchart*
Chapter 3: Methodology

The previous model was adopted from Chierichetti et al. [8]. The difference is that the interval between two consecutive clicks to open in a new tab is considered as 0. However, for this project the time between events that cause network activity is important. Thus, it is assumed here that since the page has interesting links, the user will spend some more time in the same page after they have clicked on a link to open in a new tab. The amount of time will be also exponentially distributed, but with a smaller average than the initial dwell time.

The probability of closing a tab is called death probability and the probability of clicking a link to open in a new tab in the background, while remaining in the same web page is called spawn probability. According to Chierichetti et al. [8] the death probability is around 0.7 and the spawn probability around 0.1, although there is great variance. The research from Huang and White [15] agrees with this spawn probability since their results show that in 88.7% of the next loaded pages are in the same tab.

While web navigating with Selenium it is possible that a lot of things can go wrong, especially considering that tabbed browsing is not yet fully supported by Selenium, so usually workarounds had to be implemented in order to incorporate it. For example, the browser can become unresponsive and a requested web page may not exist or result to an error page. In these cases and in other unexpected situations, a real life user would be able to respond accordingly, but Selenium being an automation tool needs a predefined behavior to handle them. For this project, every time an unexpected problem occurs Selenium will quit its current action and start the browsing procedure from the beginning, by requesting a new initial web page. That is step (1) in the above algorithm. Depending on the problem, Selenium may need to restart the browser and thus start again with an empty tab queue or just load a new page on the current tab, thus leaving the rest of the tab queue intact.

As mentioned before, some implementation details have been left out from the flowchart in Figure 8. That includes the termination condition after a certain number of main loops and the Selenium reaction after unexpected events.
4. Ground Truth Evaluation

Apart of being one of the most important parts of this project, determining the ground truth is one of the most complex parts as well. The reason for that is the great interactivity the modern web pages offer to their users. As discussed before, this has opened up a lot of possibilities for user actions and it makes it difficult not only to capture all the actions a certain user performs on a web page, but also to connect these actions to the generated HTTP traffic.

This can be illustrated by an example. Assuming a web page implements infinite scrolling and push notifications, then even if the scroll down event was captured, there could not be a way to know if the newly generated HTTP traffic was because of requesting new content by the user scrolling down or because the server has sent push notifications at the same time coincidentally. Therefore, it would be ambiguous if the newly generated HTTP traffic should be grouped as a new user action or belonging to the previous action that loaded the specific page. An example of such a page is the popular Twitter social platform.

Furthermore, it is possible that HTTP requests from a previous page will continue to be generated even after a new page is requested. There are two cases where this can happen. The first case is when a new page is requested while the previous one is still being constructed. This is due to how Firefox works internally. Since the time needed to build a web page by Firefox is far less than the time it takes a user to see the content of a web page and navigate somewhere else, this case does not occur often. This is assumed under normal network congestion and with a modern connection speed.

The second case is when a web page sends HTTP requests after the unload event was triggered, which means after a new web page was requested. These requests are often directed to web bugs and are usually related to tracking services, either of the unloading web page or of a third party entity. For example, an ad serving website that distributes ads to third party web pages might want to know how long an ad is viewed on a specific web page. Then, it can trigger an HTTP request back to the ad server, when the web page is unloaded. This request would include the time of unloading.

Another way of sending information back to a server is with the Beacon technique\(^3\). The Beacon technique uses an asynchronous HTTP request that is triggered by the unload event of a web page and is performed by the browser, when it is not busy. This way, there is no delay of the unloading procedure of the current page by requesting new objects and hence, the performance of the next navigation is not affected.

These kind of HTTP requests and their corresponding responses will be referred to as escaping traffic and they are wrongly grouped as traffic from the new page even though they belong to the previous page. They are a cause of imprecision for the Firefox extension. It should be highlighted here that the escaping traffic only affects the grouping abilities of the Firefox extension and not the user action identification capability.

The following sections demonstrate how the Firefox extension deals with escaping traffic, as well as the user actions that can be identified. Next an elementary quantitative and qualitative analysis of escaping traffic is presented.

4.1. Firefox Extension Variations

Since exact precision in the ground truth produced by the Firefox extension was not possible, different variations of the extension were devised. These variations have different accuracy regarding matching the HTTP traffic to user actions and different recognition ability regarding capturing all the user actions. Three such variations and their characteristics are presented below.

Version 1 is based on the page load event and it can only handle basic web navigation. Typing a URL in the address bar of the browser and clicking links that open new web pages are recognizable actions.

---

\(^3\) Beacon: http://www.w3.org/TR/Beacon
Chapter 4: Ground Truth Evaluation

However, actions like clicking links or other web elements that will generate traffic but will not load a new web page are not going to be recognized as new actions and their traffic will be grouped with the previous action’s traffic.

Version 2 is not dependent on a page load to recognize an action, so it can distinguish more actions. Clicking on links that only alter a part of the web page can be recognized as new actions. However, more advanced interaction with the web page, like scrolling through it or clicking buttons will not be captured even if it results in HTTP traffic.

Version 3 is based on Version 2 with additional support of the back and forward button. In the previous versions, clicking on the back/forward button will not account for a new action. It should be noted here that the support extends to moving back or forward in the browser’s history in any way, for example even if keyboard shortcuts are used.

For a better understanding of the extension variations, an example traffic timeline is shown in Figure 9. This timeline shows two web pages being loaded consecutively. While the first page is being loaded, a user event forces the second page to start loading. The user event can be any of the supported user events: click on a link, type a URL in the address bar and navigate through the browser’s history. The first request event refers to the initial HTTP request that is generated by the browser in order to load the new web page. The page load event is a Firefox internal event that shows that a new DOM tree is going to be constructed and a new page is indeed going to be shown to the user. The traffic belonging to the new page between the first request event and the page load event is actually redirections from the user requested page to the actual page that is going to be shown. It should be noted here that the above timeline is completed in an average of a few hundreds of milliseconds for a modern PC with modern Internet connection.

In all three variations the traffic belonging to the previous page that comes after the page load event is not distinguishable from the traffic of the new page. This traffic will always be escaping traffic. On the other hand, all three variations will recognize that the traffic before the first request event belongs to the previous page with one exception: in Variation 3 and only when the user event is stepping through the browser’s history all the traffic after the user event will be matched as belonging to the new page.

The main difference between Variation 1 and 2 is that Variation 1 is able to distinguish between previous and new page for the traffic between the first request event and the page load event. Variation 2, though, matches all traffic after the first request event as belonging to the new page.

The variations of the extension exist because of the variations of the counter manager. For example, the difference between Variation 1 and 2 lies in the fact that the counter manager in Variation 2 advances the counter after the first request event, while in Variation 1 it conditionally chooses counter for the HTTP traffic during the time between first request event and page load event, depending on whether the traffic comes from a redirection.

4.2. Extensions Comparison

Since the escaping traffic cannot be totally eliminated it would be useful to perform a quantitative analysis. Escaping traffic appears only when loading consecutively two pages in the same tab.
Considering that any HTTP request can potentially be escaping traffic, it is difficult to recognize and count which HTTP requests belong to the first and which to the second web page.

In this direction, the following test is run: a website is requested and immediately after it has loaded an empty page is requested. This procedure is repeated for a number of websites with the help of Selenium. Since an empty page does not create any HTTP traffic, all the HTTP traffic found after the empty page request belongs to the previous web page and is basically escaping traffic.

It could be argued here that for this test it would be better to interrupt the page load in its beginning. That is because there are usually more objects arriving in the beginning of a page load than when the page is fully loaded and thus the worst case scenario would have been examined. However, there are three reasons against it.

First is that Selenium does not support arbitrary interruption of a page load. When a command for a page load is issued from Selenium, all the static content of the page has to be downloaded before Selenium can continue with the next command, which in this case is an empty page load command. The second reason is more important though. Web pages do not load with the same speed. The round trip time for the HTTP traffic for a Swedish website is much less than a Chinese one, due to the physical location of their servers (assuming the client is in Sweden). Thus, the beginning of a web page load from the two sites will be two very different times. Statically determining the beginning of the page load will cause that the two web pages will be in different phases of their actual loading. The last reason has to do with the fact that it is not sure that most of the objects will arrive at the beginning of the page load. It could happen that initially only a stylesheet and a script will be requested and when their responses arrive to the browser and the script is run, only then a lot of new objects will be requested, generating a lot of HTTP traffic. In other words, the way a web page is loaded is specific to the web page.

The test was performed on the worldwide top 600 websites as found from the alexa.com analytics website\(^4\). Using only popular websites does not bias the results, since according to Butkiewicz et al. [22], the popularity of a website is not an indicator of its complexity. Due to instabilities of Selenium, 448 of them were included in Table 2. These websites were common for Variations 1 and 2. The percentage shown in the table represents the amount of escaping traffic compared to the total amount of HTTP traffic for the course of the test.

There are two things that should be mentioned regarding Variation 3. First, the test for it was slightly different than the other variations. In this case an empty page was initially loaded, then a selected web page was loaded and then the browser back button was clicked. Of course there will be not traffic grouped with the first empty page, but there will be some traffic grouped with the second empty page because of the loading of the regular web page. This traffic is the escaping traffic.

Secondly, the escaping traffic calculated this way refers only to when a step back in the browser history is taken. For link clicking/URL entering navigation, the escaping traffic will follow the pattern of Variation 2, since Variation 3 is based on it. Therefore, the percentage in parenthesis shown in Table 2 concerns only stepping through the browser’s history and should not be compared to the percentages of the other variations.

According to Obendorf et al. [12] 63% of web navigation is link clicking or URL entering, while 15% is navigating through the browser’s history. The rest relates to user actions that are not supported by any variation of the extension, for example form submission. Since some actions are excluded from this project, only the relationship between the frequency of link clicking/URL entering and browser’s history navigation is important. Link clicking/URL entering happens 4.2 times more often than browser’s history navigation. So, if only these two types of user actions were to be considered, 76% would be link clicking/URL entering and 24% browser’s history navigation.

Using the aforementioned frequencies the percentage of escaping traffic over the total amount of HTTP traffic for Variation 3 is estimated as: (amount of escaping traffic when link clicking or URL entering)*(frequency of link clicking or URL entering) + (amount of escaping traffic when navigating in browser history)*(frequency of navigating in browser history) = 0.74*0.76 + 1.27*0.24 = 0.87%.

\(^4\) Alexa analytics: alexa.com
Chapter 4: Ground Truth Evaluation

<table>
<thead>
<tr>
<th>User action handled:</th>
<th>Firefox extension</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Variation 1</td>
</tr>
<tr>
<td>Typing URL</td>
<td>✓</td>
</tr>
<tr>
<td>Clicking links that load new page</td>
<td>✓</td>
</tr>
<tr>
<td>Clicking links that load new content in current page</td>
<td>✗</td>
</tr>
<tr>
<td>Stepping through the browser’s history</td>
<td>✗</td>
</tr>
<tr>
<td>Escaping traffic</td>
<td>0.45%</td>
</tr>
</tbody>
</table>

*Table 2. Extension variations comparison*

The test verifies what was intuitively explained before: being able to recognize more user actions reduces the ability of HTTP grouping in the extension and thus reduces the accuracy of the produced ground truth.

### 4.3. Ad/Tracking Related HTTP Traffic

In an attempt to further examine the escaping traffic the discussion in section 2.6 should be advised. Most of the objects that arrive towards the end of a page load do not belong to the same domain as the web page. That makes it more possible to belong to ad and/or tracking services, which usually belong to external websites.

Figure 10 shows the loading of the same page with and without blocking ad/tracking services. More specifically, the Adblock Plus Firefox extension was used to block ad and tracking related URLs⁵. Adblock Plus is the most popular Firefox extension for blocking unwanted content on the web and is based on lists of filters. Here the EasyList and EasyPrivacy lists were used⁶. EasyList is mainly used for blocking ads, while EasyPrivacy for blocking tracking services. As before the lower graph shows the HTTP requests that belong to the dominant domains.

![Fig. 10. Loading huffingtonpost.com with and without Adblock Plus](image)

---

⁵ Adblock Plus: adblockplus.org
⁶ EasyList: easylist.adblockplus.org/en/
The impact of the ad/tracking related objects is clearly obvious as the page loading progresses. After some initial loading, with Adblock Plus enabled, the graph of all domains follows the graph of the dominant domains, while being around 50 requests elevated. On the other hand, with Adblock disabled, the total number keeps increasing, while the number of HTTP requests addressed to the dominant domains stays constant. Furthermore, the amount of requests towards the dominant domains is the same with and without Adblock. All these can justify the argument that the escaping traffic which occurs at the end of a page load is closely related to ad/tracking services.

Another way of seeing this is with the following test. As before, a web page was loaded and then immediately an empty page was requested. This time though, the test was run not for different variations, but with different Firefox settings regarding blocking ad and tracking related URLs. In both cases Variation 2 was used. Again, the traffic belonging to the empty page is escaping traffic, considering that an empty page does not generate any network activity. Since it is difficult to know from before which web pages will cause escaping traffic, the test was repeated for 200 different websites and the amount of escaping traffic was summed for all these pages.

After the tests were performed, the number of escaping HTTP requests as well as the total number of HTTP requests was counted for the two cases. For comparison and due to the fact that Adblock Plus will eliminate HTTP traffic in general and not just escaping traffic, the percentage of escaping traffic over the total amount of traffic is also calculated and included in Table 3.

<table>
<thead>
<tr>
<th></th>
<th>without Adblock Plus</th>
<th>with Adblock plus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Escaping traffic (absolute)</td>
<td>151</td>
<td>51</td>
</tr>
<tr>
<td>Total number of HTTP traffic</td>
<td>14,944</td>
<td>11,246</td>
</tr>
<tr>
<td>Escaping traffic (relative)</td>
<td>1.01%</td>
<td>0.45%</td>
</tr>
</tbody>
</table>

Table 3. The effect of ad/tracking services

This shows once more that the escaping traffic is closely related to ad/tracking services which lowers its significance regarding the ground truth. By using a blocking mechanism the amount of escaping traffic was reduced 55%. In other words, the ground truth accuracy can greatly be increased by using an ad/tracking blocking mechanism.

To further investigate the nature of the escaping traffic, 214 HTTP requests that belong to escaping traffic from the previous tests were manually examined by opening them separately in a browser window. 58% of them result in the fetching of an image of size 1x1 pixel. This kind of images relate to web bugs and are a common mechanism for web tracking. Moreover, another 12% correspond to JavaScript objects which are often used for user tracking. This comes in accordance to the research of Krishnamurthy and Wills [16] regarding extraneous content on web pages. In their results, 94% of the ad/tracking related objects consist of either JavaScript objects or images.

In summary, the main results of this preliminary quantitative and qualitative analysis are that the Firefox extension introduces approximately 1% of escaping traffic and that around 50% of it is related to ad/tracking services. The amount of escaping traffic seems very low for the purposes of this project and will be ignored in further discussions. The fact that a big part of the escaping traffic can also be categorized further strengthens the decision to ignore it.

4.4. Browser Related HTTP Traffic

Some of the HTTP traffic cannot be matched to any tab. This traffic is generated by the browser itself and not by the loading of a user requested web page. It reflects the modern browsers’ additional functionality. Some common categories of unmatched traffic are next described.
Safebrowsing: During a browser session Firefox will communicate to a Google service called Safe Browsing in order to obtain lists of suspected phishing and malware pages. This can happen multiple times depending on the duration of the session. This kind of traffic can easily be recognized by checking the domain name of the URL of the HTTP traffic, which will be something like: safebrowsing.google.com. Firefox gives the option of disabling this feature.

OCSP: The Online Certificate Status Protocol is used online for obtaining the revocation status of an X.509 public key certificate. In order to check that the digital certificate of a web page has not been revoked the browser will communicate with OCSP servers. This kind of traffic can usually also be easily recognized because it includes the string ocsp in its URL. Additionally, its content-type header field is “application/ocsp-request” or “application/ocsp-response”. OCSP related traffic is not of a great importance for this project since it is entirely connected with web security. Moreover, Google Chrome has even disabled this kind of mechanism.

Favicon: A favicon is a file containing a small icon used by browsers to display this icon next to the title of a web page. The favicon is not an embedded object of a web page and the browser will look for it at a fixed location at the server of a website, even if the website’s creators have not created one. Since the favicon is not directly related to a web page’s content, the traffic it creates is unmatched. Favicon files can be usually recognized by one of the strings ico, icon, favico or favicon in their URLs.

Since this kind of traffic does not constitute direct user actions, the caused imprecision is only affecting the grouping abilities of the ground truth and not the user action identification. By visiting 560 web pages in a Tabbed Browsing manner (as it will be described in section 5.3), it was found that 4.14% of all the recorded traffic is unmatched traffic. 92% of these unmatched traffic belongs to one of the aforementioned categories.
5. Proxy-log User Action Identification

The aim of this project is to identify user actions from within a proxy log. A user action creates a group of HTTP requests as discussed before. The first HTTP request of a group is the identifier of the group, since it is the most connected with the user action. For example, if the user action is typing www.example.com in the address bar of a browser, then the first HTTP request of the generated HTTP traffic will contain the URL www.example.com, while the rest of the traffic will have URLs in the form of: www.example.com/img/pic.jpg, img.example.com/pic.jpg or even stats.adserver.com/stats.js. Thus, the user action and the first HTTP request of the generated traffic are closely related. For the most part of this chapter, the term user action will be used instead of first HTTP request corresponding to a user action.

The user actions are identified by applying the proposed policies on the proxy logs. These policies are based on heuristics and they can be grouped into two categories. The time-based policies take into consideration when and how often HTTP requests are generated, while the HTTP-specific policies use either the URL or one of the HTTP header fields of an HTTP request.

5.1. Time-based Policies

*Previous HTTP request: Previous policy*

The first policy for distinguishing user actions from the proxy log is time-based. An HTTP request is considered a user action if the difference between its timestamp and the timestamp of the previous HTTP request is greater than a threshold. This policy will be referred to as the Previous policy, because for every HTTP request the time distance from the previous HTTP request is of importance. Its threshold will be referred as the Previous Window Threshold. The Previous policy is also used by Liu et al. [17] as the first stage of their algorithm for user-click identification.

The rationale behind this policy is that usually the interval between two consecutive user actions is far larger than the interval between consecutive HTTP requests for objects within a web page. Nonetheless, due to various techniques used by the author of a web page and/or the proximity of corresponding web servers and client, it can happen that HTTP requests for objects are generated in intervals that are comparable to the time it takes a user to request the next page. This would give false positives. Moreover, if the time between two user actions is very short then the second action would not be recognized by this policy. These are cases of false negatives.

This first policy constitutes the basis for the rest of the policies. This means that after this policy the proxy log is grouped by potential user actions. Each user action is followed by its consecutive HTTP requests. Additional policies can be applied afterwards. These policies can either eliminate specific suggested user actions or look for a better match for a user action among the consecutive HTTP requests.

*Path-based repetition of HTTP requests: Repetition policy*

Before presenting the next policy, a brief description of a URL’s structure will help clarify some terminology. A typical URL of an HTTP request has the following parts: hostname, path, query, fragment as shown in Figure 11. The hostname represents the server that the HTTP connection is made to, the path shows the location of the requested resource in the server’s file system, the query carries various useful parameters for the particular resource and the fragment indicates a specific location within the requested resource. Only the hostname is necessary to establish an HTTP connection. It should be noted here that the term path often excludes the hostname in the bibliography, but for this project the term path is including it.
From observation of HTTP traffic, it can be concluded that HTTP requests that belong to embedded objects in web pages are often directed to URLs with the same path, while HTTP requests that constitute user actions have usually unique URL paths. For example, a request for a page could have the URL: http://www.example.com/ and some of the consecutive HTTP requests could be: http://www.example.com/load.php?module=site and http://www.example.com/load.php?module=gadget. This is especially common with scripts within a web page.

Furthermore, a lot of web pages need to update their content or usually part of their content. This update generates HTTP traffic that is not a result of a user action. This HTTP traffic occurs well after the page has loaded, so they are suggested by the Previous-policy. By observing client logs, it can be inferred that for every web page this kind of generated HTTP traffic is usually directed to a specific resource and the update is repeated with a constant frequency.

This observation has led to the idea of eliminating regularly repeated HTTP requests from user actions. In order to decide if a set of consecutive HTTP requests directed to a specific resource shows a regular repetition behavior, the variation coefficient of the timestamp differences of these requests is used. The variation coefficient is calculated by dividing the standard deviation of a set of values to the mean value of the set. A low variation coefficient means that the set is highly regular. In other words, if the variation coefficient is less than a threshold, referred to as the Regularity Threshold, then these HTTP requests have a regular behavior and should be eliminated from the user actions.

In order to eliminate regularly repeated HTTP requests there is need to create a list with all the different requested URL paths and each entry of this list is another list with the timestamps when this URL path has appeared. Then, if a regular repetition is discovered for a subset of consecutive timestamps, then the corresponding HTTP requests will be filtered out in the selection of user actions. An initial value for the size of the subset is 4.

When determining a suitable Regularity Threshold an interesting observation was made. In Figure 12, the F1 score of the Previous policy with and without elimination of regularly repeated URL paths is plotted for various values of the Regularity Threshold. These results were obtained by browsing 500 websites (following the Simple Browsing model with average dwell time of 20 s, as it will be described in section 5.3). Although, lower values of the Regularity Threshold mean a highly regular repetition of URL paths, it can be seen that the F1 score improves further when even barely regular repetitive URL paths are eliminated. This shows that it would be more effective if even non-regularly repeated URL paths are filtered out. The observation that a URL path is repeated is enough to make it unsuitable as a user action.
Therefore, it is enough that HTTP requests that have URL paths which are repeated more than a certain threshold should be rejected as user actions. This leads to the second proposed policy which will be referred to as the Repetition policy and its threshold as the Repetition Threshold.

This policy is implemented by keeping track of all the different URL paths and how many times each one has appeared. If a suggested user action by the Previous policy has appeared more than the Repetition Threshold then this HTTP request is rejected and the next HTTP request becomes a user action candidate. It is obvious that this policy is less effective in the beginning of a proxy log, since all the HTTP requests in the beginning seem to have appeared only once and therefore cannot be eliminated by the Repetition policy.

Next HTTP request: Next policy

For this third policy, in order for an HTTP request to be a user action, it must not be followed by any HTTP requests for a specific time period. This will be the Next policy, since the user action condition utilizes the period until the next HTTP request. The aforementioned time period will be referred to as Next Window Threshold.

An intuitive explanation of this policy lies in the fact that the HTTP traffic generated by a web page load has a bursty behavior. As discussed in Chapter 2, when a user requests a page, an initial HTTP request is generated by the browser for the specific page, followed by bursts of HTTP requests for the rest of the objects in that page (CSS files, scripts, images and other). Therefore, for any HTTP request, if there are more HTTP requests within the Next Window Threshold, it means that this HTTP request is a part of a burst. On the contrary, if the Next Window Threshold is empty of requests, then the HTTP request has a good chance of being an HTTP request for a web page.

The way this policy is implemented is that for every suggested user action from the Previous policy, every HTTP request in that user action group and its following HTTP request are checked against the Next policy. If they pass the check, then this HTTP is a potential user action. If they fail, then the policy is checked with the following HTTP request and its following HTTP request. This can continue until there are no more HTTP requests in that group.

This policy is also vulnerable to the proximity of web server and client, but in an opposite manner. With the Previous policy, a faraway server will cause more HTTP requests to be wrongly recognized as user actions (false positives), while with the Next policy, a nearby server will result in missing user actions (false negatives), because the HTTP bursts will be close to each other.

Since the Next policy needs not only information about an HTTP request and its past, but also about the next HTTP request, this policy will be applied only after the other policies have been applied.
Chapter 5: Proxy-log User Action Identification

**Time-based policies**

With the term time-based policy a combination of Previous, Repetition and Next policy is implied. Figure 13a-c show a graphical representation of the identification and grouping abilities of various time-based policies. The scenario here is two consecutive user actions. The user actions correspond to requesting page A and page B. The arrows represent HTTP requests caused by the loading of the pages. The larger arrows represent the HTTP requests that initiate the loading of the pages. The first HTTP request of every group is a suggested user action by the policy.

In Figure 13b the repetitive HTTP request previously suggested as user action 2 in Figure 13a is now eliminated by the Repetition policy. The elimination of false positives will lead to higher precision. By adding the Next policy in Figure 13c, one more false positive is eliminated and a correct user action is suggested. In this case both precision and recall will increase. The case where HTTP traffic is generated right before a new page is loaded often belongs to ad/tracking services as discussed in section 4.3.

![Graphical representation of the time-based policies](image)

---

**5.2. HTTP-specific Policies**

**URL-based policy: URL policy**

The Previous policy takes into consideration the logged time of the HTTP requests. In addition, the Repetition policy utilizes also their URLs. The next proposed policy relies on the idea that HTTP requests for certain resources usually do not constitute user actions. These resources are favicons, ocsp transactions and ad/tracking related services. As discussed in section 4.4 these HTTP requests are either initiated by the browser itself or by a web page, but not from a user. Nonetheless, it is possible that a user requests these resources willingly, although extremely rare for an average user.

This URL-based policy checks every requested URL and if terms like “favico”, “.ico”, “ocsp”, “googlesyndication”, “google-analytics”, “doubleclick”, “2o7” are found, then this HTTP request is filtered out from the user action suggestion process. The ad/tracking related URLs are well known and frequently found services [21]. It is obvious that the URL policy needs another policy to be previously applied.
Content-type-based policy: Content policy

This next policy is similar to the previous one, but instead of filtering out HTTP requests based on their URL, it is based on the content-type HTTP header field. This field reveals the type of data an HTTP response contains and the names of these types are registered by Internet Assigned Numbers Authority (IANA)\(^7\). They are also known as Internet media types or MIME types. The most frequent MIME types are text, image and application and some frequently encountered subtypes include html, javascript, css, and png.

User actions have often text/html as their content-type, while */javascript, text/css and image/* are usually embedded objects of a web page. In the light of this, the Content policy constitutes of filtering out every HTTP request that is considered to be an embedded object and keeping only HTTP requests with text/html as their content-type.

Observation of the recorded traffic at the proxy revealed that there are copious HTTP transactions which are labeled with the text/html content-type. Therefore, applying only the Content policy would result in extremely low precision. For this reason, the Content policy needs to be treated as a second step filter, meaning that as a first step another policy should be applied, as for example a time-based policy. When the first step policy has decreased substantially the number of suggested user actions, the Content policy will filter out false positives, improving, thus, the overall recall.

Referer policy

In an attempt to use the referer field technique in proxy logs the following simple idea could be used: while processing the proxy log, every distinct URL that is found in the referer field is a user action. This policy, though, has a few problems. Some elements in web pages generate HTTP traffic on their own and this traffic has the elements as referer. This causes false positives. Moreover, when a requested page leads to redirection then the following traffic has the redirected page as referer. In this project, a user action is what a user has clicked on and not what was finally loaded on their browser. Thus, the redirected page that will be suggested by the algorithm is a false positive in terms of this project. Finally, due to caching in the browser it is possible that a user request goes to proxy, but since the server answers that there is no change of content, no more HTTP requests for embedded objects (which would contain a referer field with that user action) will go to the proxy, resulting to a false negative for that cached user action.

Table 4 shows quantitatively why the Referer policy is not a good fit. The number of actions present in the client, the number of actions suggested by the Referer policy, the matching user actions on client and server and their evaluation metrics are shown.

<table>
<thead>
<tr>
<th>Client</th>
<th>Proxy</th>
<th>Match</th>
<th>Precision</th>
<th>Recall</th>
<th>F1 score</th>
</tr>
</thead>
<tbody>
<tr>
<td>505</td>
<td>1338</td>
<td>361</td>
<td>0.27</td>
<td>0.71</td>
<td>0.39</td>
</tr>
</tbody>
</table>

Table 4. Evaluation of the Referer policy

There is also another problem with the Referer policy that is independent of the methodology followed by this project. Due to privacy concerns the HTTP Referer field is often left empty or altered by the browsers in cases when users do not want the new requested page to know the previous visited page. Since the aim of this project is to track user behavior from proxy logs and not from client logs where it would be possible to instruct the browser to always send the correct referrer in the HTTP header, the referer field will be ignored for this project.

\(^7\) Media types: http://www.iana.org/assignments/media-types/media-types.xhtml
6. Evaluation Results

The primary evaluation measures are the precision and recall of each policy. Precision is the number of correct user actions found in the proxy log out of the total number of suggested user actions in the proxy log. Recall is the number of correct user actions found in the proxy log out of the number of total user actions in the ground truth. Precision and recall are often used in data mining. Another measure from data mining that incorporates both precision and recall is the F1 score and is calculated by the following formula:

\[ F1 = \frac{2 \times (\text{Precision} \times \text{Recall})}{\text{Precision} + \text{Recall}}. \]

After gathering proxy and client logs with the help of Selenium, the different policies are applied to the proxy logs. This results in a list of user actions. The ground truth is extracted from the client logs and is also a list of user actions. The evaluation of the different policies is basically the comparison of these two lists.

The following results refer to browsing 500 web pages. The behavior model that was followed is the Simple Browsing model (no use of multiple tabs). These web pages are either picked randomly out of a pool of web pages or are the result of following links. It should be noted here that since the web pages are picked randomly and the links are also picked randomly, the resulting 500 web pages may not be unique. The pool out of which web pages are selected is filled with the one thousand most popular web pages as presented by alexa.com, excluding pages that are based on HTTPS and Chinese websites. As discussed before, HTTPS traffic is not supported by Squid and it was observed that Chinese sites cause problems to Selenium resulting it to crash. Moreover, the average dwell time was 20 s.

Selenium was used together with Variation 2 of the Firefox extension. Variation 2 recognizes typing a URL into the address bar and clicking on any type of link. Being able to recognize any type of link clicking is important because Selenium cannot identify between links that load a new page and links that load part of a page. Moreover, due to caching that is performed by Firefox, using the browser’s history would result in fewer objects to be requested by the network and thus through the proxy server. For this reason, navigating through the browser’s history will not be included at this point and use of Variation 3 is not necessary.

The simulation is run by only one client machine that is instrumented to use the Squid proxy. Dealing with more than one user is a simple addition on the proxy software. It just needs to record the source IP address of each HTTP transaction and possibly the source port number as well. Since distinguishing among users is a trivial task and the policies are applied independently on each user, tests run from only one user are sufficient for the accuracy evaluation of the proposed policies.

By repeating the simulation 5 times, the average and the standard deviation of the evaluation metrics are obtained. Table 5 presents the results with respect to the following parameter values:

- Previous Window Threshold = 2 s,
- Repetition Threshold = 5, and
- Next Window Threshold = 20 ms.

The number in the parenthesis is the standard deviation. Figure 14 plots the precision with respect to recall for the various policies and shows clearly the improvement achieved by each policy. Figure 14 was obtained by varying the Previous Window Threshold from 500 ms to 10,000 ms with a step of 500 ms. For Previous Window Threshold equal to 500ms the highest recall and lowest precision are reached for each policy. For the rest of the project, all of the graphs that present precision in terms of recall are obtained the same way.

Starting with only the Previous policy and adding the rest of the time-based policies, it is possible to reach better precision while having the same recall. Increase in precision of up to 10% with each additional time-based policy is possible. Similarly, by using the same combination of time-based policies and by adding the URL or Content policy, further improvement of precision can be achieved while
Chapter 6: Evaluation Results

maintaining the same recall. In other words, the additional policies are able to trim away a good deal of false positives without affecting the correct matches.

Although, the URL- and Content-based policies accomplish the best results, it should be kept in mind that the URL and the content-type information about an HTTP transaction are not always available at a proxy level recording point. The proxy should have access to the headers of the HTTP message.

<table>
<thead>
<tr>
<th>Method</th>
<th>Recall</th>
<th>Precision</th>
<th>F1 score</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time-based only</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Previous+Repetition+Next</td>
<td>0.8251</td>
<td>0.7822</td>
<td>0.8030</td>
</tr>
<tr>
<td></td>
<td>(0.0181)</td>
<td>(0.0181)</td>
<td>(0.0143)</td>
</tr>
<tr>
<td>Previous+Repetition</td>
<td>0.8290</td>
<td>0.6962</td>
<td>0.7568</td>
</tr>
<tr>
<td></td>
<td>(0.0196)</td>
<td>(0.0225)</td>
<td>(0.0213)</td>
</tr>
<tr>
<td>Previous</td>
<td>0.8067</td>
<td>0.5969</td>
<td>0.6860</td>
</tr>
<tr>
<td></td>
<td>(0.0307)</td>
<td>(0.0391)</td>
<td>(0.0367)</td>
</tr>
<tr>
<td><strong>Time-based and URL-based</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Previous+Repetition+Next</td>
<td>0.8259</td>
<td>0.8251</td>
<td>0.8254</td>
</tr>
<tr>
<td></td>
<td>(0.0156)</td>
<td>(0.0164)</td>
<td>(0.0123)</td>
</tr>
<tr>
<td>Previous+Repetition</td>
<td>0.8303</td>
<td>0.7332</td>
<td>0.7787</td>
</tr>
<tr>
<td></td>
<td>(0.0166)</td>
<td>(0.0250)</td>
<td>(0.0209)</td>
</tr>
<tr>
<td>Previous</td>
<td>0.8095</td>
<td>0.6437</td>
<td>0.7170</td>
</tr>
<tr>
<td></td>
<td>(0.0290)</td>
<td>(0.0438)</td>
<td>(0.0385)</td>
</tr>
<tr>
<td><strong>Time-based and Content-based</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Previous+Repetition+Next</td>
<td>0.8287</td>
<td>0.8992</td>
<td>0.8625</td>
</tr>
<tr>
<td></td>
<td>(0.0158)</td>
<td>(0.0188)</td>
<td>(0.0162)</td>
</tr>
<tr>
<td>Previous+Repetition</td>
<td>0.8339</td>
<td>0.8271</td>
<td>0.8304</td>
</tr>
<tr>
<td></td>
<td>(0.0167)</td>
<td>(0.0333)</td>
<td>(0.0247)</td>
</tr>
<tr>
<td>Previous</td>
<td>0.8191</td>
<td>0.7883</td>
<td>0.8032</td>
</tr>
<tr>
<td></td>
<td>(0.0303)</td>
<td>(0.0515)</td>
<td>(0.0409)</td>
</tr>
</tbody>
</table>

Table 5. Evaluation metrics for the proposed policies
Chapter 6: Evaluation Results

6.1. Number of Suggested User Actions

The previous results aim to show how accurate the proposed policies are compared to the ground truth. In this section the number of suggested user actions by each policy is discussed and plotted in Figure 15. The suggested user actions consist of a subset of the actual user actions and other HTTP requests that are not user actions.

Using the same traces as before, it is found that the loading of 500 web pages generates on average 29,506 HTTP requests. Applying, for instance, the Previous+Repetition+Next policies results in only 528 suggested user actions. This means that out of a big pool of data, the particular policy is able to suggest only a tremendously small fraction of requests. As shown in Figure 15 the same is true about the other policies as well, with respect to the selected tuning parameters. In all cases, the applied policy is able to avoid more than 97% of the generated HTTP requests, as only between 450 and 680 of the total 29,506 HTTP requests are suggested as user actions.

Figure 15 also reveals the potential for improvement of each policy. Every additional policy is able to eliminate more false positives and thus achieve better precision. The number of matches stays almost constant for all the policies which means that recall is almost the same. As a matter of fact, the number of

![Fig. 14. Comparison of the proposed policies](image)

![Fig. 15. Number of suggested user actions in comparison with actual user actions for each proposed policy](image)
matches gets slightly higher as more policies are applied. For example, with the Previous policy there are on average 404 matches, while with the Previous+Repetition+Next+Content policy there are 415 matches.

6.2. Time-based Policies Evaluation

Figures 16-21 demonstrate the effect of the various parameters on the time-based policies. More specifically, Figures 16, 18 and 20 show the effect on the F1 score and Figures 17, 19 and 21 show the effect on precision and recall. The procedure for this part of the evaluation is to vary one of the tuning parameters while keeping constant all the others. In order to be uniform with the previous results, when a parameter is kept constant then it has the same value as discussed in the context of Figure 14. The URL and Content policy were not applied for this part.

In Figure 16 the F1 score is plotted as a function of the Previous Window Threshold with fixed values for the Repetition and the Next Window Thresholds. The three curves represent the three time-based policies. Independently of the policy, the best values for the Previous Window Threshold are between 0.5 and 10 s with a peak around 2 s. Smaller values of Previous Window Threshold have the effect that more user actions are suggested. However, the more advanced time-based policies will be able to eliminate a lot of these user actions. That is the reason why the more advanced time-based policies favor slightly the smaller Previous Window Threshold values.

Figure 18 summarizes the effect of the Repetition Threshold on the F1 score. Here, the Previous and Next Window Thresholds are fixed. It is obvious that the peak value of the F1 score is achieved with the Repetition Threshold at 5. The Previous policy does not use the Repetition Threshold and so its F1 score is not affected by it. These curves are approximations since the Repetition Threshold represents a natural number and not time as in Figures 16 and 20. Only the highlighted points have calculated values.

The effect of the Next Window Threshold is shown in Figure 20. Values between 10 and 50 ms achieve the highest F1 score. It should be noted here that values more than 100 ms drastically deteriorate the performance. For example, when the Next Window Threshold is 120 ms the F1 score is lower than when the Next policy is not applied at all. The reason is that a lot of correct user actions that were proposed by the other policies, will be wrongly eliminated.

In Figure 17, all three time-based policies are applied together and the Previous Window Threshold is varied while the Repetition and Next Window Thresholds are fixed. In general, it is important to achieve both high precision and high recall. Thus, the intersection of the two curves reveals the best Previous Window Threshold. Again here it can be seen that a value of 2 s for the Previous Window Threshold achieves this.

Precision/Recall of all three time-based policies are plotted as a function of the Repetition Threshold at Figure 19. Previous and Next Window Thresholds are fixed in this case. The two curves intersect at a value around 4 which gives the best precision-recall tradeoff. These curves are approximations since the Repetition Threshold is actually taking only integer values.

Finally, in Figure 21 precision and recall are plotted as a function of the Next Window Threshold with the Previous Window and Repetition Thresholds being fixed. Figure 21 agrees with the observations from Figure 20 in the sense that the best values for the Next Window Threshold are between 10 and 50 ms.

From these figures it can be gathered that the following values achieve scores close to the optimal:

- Previous Window Threshold = 2 s,
- Repetition Threshold = 5, and
- Next Window Threshold = 20 ms.

Therefore, for the rest of this document and if not otherwise stated, these values will be used when the time-based policies are applied.
Chapter 6: Evaluation Results

**Fig. 16.** Effect of the Previous Window Threshold on F1 score

**Fig. 17.** Effect of the Previous Window Threshold on Precision/Recall when all time-based policies are applied

**Fig. 18.** Effect of the Repetition Threshold on F1 score

**Fig. 19.** Effect of the Repetition Threshold on Precision/Recall when all time-based policies are applied

**Fig. 20.** Effect of the Next Window Threshold on F1 score

**Fig. 21.** Effect of the Next window threshold on Precision/Recall when all time-based policies are applied
6.3. Effect of Average Dwell Time

Since time-based policies constitute a major part of this project, it would be interesting to see how they are affected by a different average dwell time. As mentioned before in section 5.3 the dwell time refers to the time a user stays on a web page and it was modeled as exponentially distributed for this project. The previous results were achieved with an average dwell time of 20 s. Figures 22-24 were obtained by following the Simple Browsing model for 500 pages each time with an average dwell time of 40 s, 30 s and 10 s respectively.

More specifically, Figures 22-24 plot precision with respect to recall when various combination of policies were applied. Applying the Repetition policy after the Previous policy increases considerably the identification accuracy of the algorithm. The addition of the Next policy further improves the accuracy, while the use of the URL policy slightly pushes the identification capabilities. Finally, the combination of time-based and Content policies achieves as always the highest accuracy. These results follow the same patterns as Figure 14, thus, it can be concluded that the effect of the policies is the same, regardless of the average dwell time.

In the cases of average dwell time of 40 s and 30 s, it can be seen that for Previous Window Threshold of 500 ms the combination of time-based and Content policies achieves lower recall and only slightly better precision than the combination of time-based and URL policies or even just the combination of time-based policies. This is not the case when the average dwell time is 20 s or 10 s, because for Previous Window Threshold equal to 500 ms recall is almost the same while precision is much higher for the Previous+Repetition+Next+Content policy. The explanation for this lies in the fact that as already seen in section 6.1 the combination of time-based and Content policies is the most aggressive policy in terms of reducing the suggested user actions. However, when dealing with longer average dwell times more of the suggested user actions are actual user actions, indicated by the higher recall values. Thus, the Previous+Repetition+Next+Content policy ends up having a negative effect on recall, rather than improving precision.
In order to directly compare the different average dwell times, Figure 25 shows the Precision/Recall results of all four average dwell times. Only the three time-based policies (Previous, Repetition and Next) were applied for Figure 25. The reason why longer average dwell times give better accuracy is that less user actions go unnoticed for the same Previous Window Threshold. Thus, the recall is generally better for longer average dwell times. Despite that difference, the results are quite similar for the four cases.

The similarity of the four cases can also be seen in Figure 26 where the F1 score is plotted with respect to the Previous Window Threshold. Again here only the three time-based policies have been applied. For up to 5 s of Previous Window Threshold the differences in F1 score are small. An observation that can be made is that longer average dwell times give sufficiently good accuracy for even larger Previous Window Thresholds. That is because most of the user actions occur anyway after longer intervals, so the Previous policy succeeds in discovering them even with larger Previous Window Thresholds. As a conclusion, without knowledge about the average dwell time, a Previous Window Threshold of 1 s to 3 s is the safest choice.

**6.4. Tabbed Browsing**

The following results are achieved with the use of the Tabbed Browsing scenario. The average dwell time for the initial page load is 20 s, while the average dwell time after clicking a link to open in a new page is 10 s. The results are presented in Figure 27 and are in complete accordance with Figure 14. This is an important conclusion since it means that the proposed policies can be used to identify user actions even in a tabbed browsing scenario.
Chapter 6: Evaluation Results

Fig. 27. Precision/Recall results for the proposed policies in a tabbed browsing scenario
7. Discussion

Although the evaluation results for the proposed policies may not seem impressive at a first glance, it should be kept in mind that these policies are very generic and can be applied to any proxy server log, since they do not need advanced and specific information to be recorded in the proxy logs. Starting and ending times of a transaction, the URL of the requested resource, as well as its type are easily and commonly obtained in a proxy log. No information about the proxy’s network location is required. Since the round trip time for the HTTP traffic is heavily affected by the proximity of proxy and server and most of the proposed policies are time-based, it could be argued that better results are possible if the proxy’s position is taken into consideration.

As an example, in their work, Arlitt et al. [25] demonstrated how proxy logs could be used in analyzing the performance of various web services and/or identifying the infrastructure of those services. The information that were extracted from their proxy logs regard application-level transactions and include start time, duration and content type. In fact, this set of information is enough for the time-based (except from the Repetition policy, where the URL path is also needed) and content-based policies that were proposed in this thesis. Therefore, a user action identification analysis could be performed on these proxy logs without any additional tuning of the existent proxies.

Moreover, the proposed policies are not computationally complex. They use either a single variable to maintain some sort of memory and compare a current HTTP request against that variable, or pattern matching. This allows for immediate application of the policies while a proxy receives the HTTP traffic. The only exception is the Next policy, since it needs information about upcoming HTTP requests as well.

The framework for validating and obtaining the evaluation measures for the identification policies is automated and can be used for large scale testing. Thus, the need of a domain expert to manually validate each policy’s results is eliminated. This is a major advantage of this project since in previous work the need of manual inspection of the identification policy results resulted in limited testing [17]. This way if more policies are proposed in the future this project’s framework will be able to easily and quickly provide evaluation results.

Throughout all of the experiments, the main results remain the same. The Next and Repetition policies always enhance the suggested user actions compared to the basic Previous policy and the Content and URL policies give the opportunity to further reduce wrongly suggested user actions and thus raise the precision of the suggestions. The generally low standard deviation is a good indicator of that.

The user actions that are not suggested by the policies do not follow a specific pattern and in fact they could have been identified, in case extreme values for the tuning parameters were used. For example, with Previous Window Threshold of 1 ms all the actual user actions would have been suggested. However, even if all of the rest of the policies were applied and even if none of the actual user actions were eliminated by them, the big amount of false positives would render precision to very low rates. Since the aim of the policies was to have high results both for precision and recall, it was preferable to sacrifice the identification of some actual user actions in order to eliminate a great deal of drag-along traffic.

Landing and non-landing pages

In order to further examine the performance of the proposed policies, a distinction between web pages was made. Landing pages are the web pages that are connected to a specific web service. Non-landing pages are the “deeper” pages that are linked from a landing page or another non-landing page. For example, bbc.com is a landing page, while bbc.com/weather is a non-landing page. In their study about web page complexity, Butkiewicz et al. [22] found that in most cases the non-landing pages are less complex than the landing ones, while there’s a smaller fraction where the non-landing page is far more complex than the landing one. The complexity here refers to the number of requests made, the number of servers contacted, the page size and the download time.
Chapter 7: Discussion

Regardless of whether the landing page is more or less complex than the non-landing one, the important result here is that there is a difference in complexity between the two types of web pages. That is the reason why it is interesting to see how the proposed policies perform in the cases of landing and non-landing pages. Figure 28 presents the results of this distinction. All client actions and suggested client actions are divided to landing and non-landing pages and their accuracy is examined separately. These results represent the average case of downloading 500 web pages for 5 times and applying all three of the time-based policies with the standard parameters. For comparison, the results when no distinction about landing and non-landing pages is made are also shown.

![Recall comparison of landing and non-landing pages](image)

It should be noted here that only recall is demonstrated in Figure 28 due to the resemblance of non-landing pages with drag-along content of a website. For instance, the URL bbc.com/script.js is much more similar to the URL of a non-landing page, like bbc.com/weather. On the other hand, the vast majority of landing pages have a URL in the form of bbc.com/ which is almost never the case for drag-along content. Therefore, the false positives of the proposed policies will only account for the non-landing pages which creates the illusion that precision is much worse for the non-landing pages. For this reason, precision and F1 score, which takes into consideration precision are intentionally left out of Figure 28. After this clarification and by looking at Figure 28, it can be concluded that the proposed policies have the same performance regardless of the level of a web page within a web service.

**HTTPS traffic**

Despite the experimentation with various websites, there was a limitation on the experiments. HTTPS traffic was avoided and ignored due to proxy software (Squid) limitations. However, popular websites, including Google, Facebook and YouTube offer their services over HTTPS by default. Even the video traffic for YouTube is over HTTPS nowadays. Therefore, it would be interesting to test how the proposed policies perform with HTTPS traffic.

Squid can be configured to intercept HTTPS traffic by using custom SSL certificates. These certificates are signed by a local authority and all the machines in the local network need to be configured to trust this local authority. This common technique for HTTPS interception is actually a Man-In-The-Middle attack and is used by default on other proxy software, like Charles proxy. Another, approach would be to keep track of TCP connections instead of HTTP transactions. In this case, port and IP numbers would also be recorded by the proxy software in order to identify different connections. The strength of the policies as they were presented would be probably worse since user actions that were served by the same connection would be indistinguishable. However, Man-In-The-Middle implementations and examination of transport-level data were outside the scope of this project.

---

Google Chrome Extension

Another limitation of the research method was the study of HTTP traffic generated exclusively by the Firefox web browser. Since the ground truth plays such an important role in the policy evaluation and the Firefox Extension was producing the ground truth, only traffic from Firefox could be analyzed. However, different browsers use different Rendering Engines, as discussed in Section 2.5 and shown in Figure 1. For example, Firefox uses Gecko and Chrome uses a project fork of WebKit. Since the Rendering Engine is responsible for the construction of the DOM tree, as well as for the Networking component and the JavaScript Interpreter, this could possibly mean differences in the way the HTTP traffic is fired and thus, differences in the identification strength of the proposed policies.

As a compensation for that limitation, a Google Chrome extension was also developed. The extension was built with the same principles regarding HTTP traffic grouping to user actions, as the Firefox extension, which is described in section 3.1 and specifically Variation 2, which is described in section 4.1. Nonetheless, tabbed browsing is not supported and it is only meant to be used together with a Selenium script and not with a human user.

The following test was run with a Google Chrome instance using the Simple Browsing Selenium model with an average dwell time of 20 s. Choosing from 500 popular web sites and links from them, 400 web pages were loaded. These are actually the same settings as in section 5.1, but with Google Chrome as browser instead of Mozilla Firefox. The Chrome Extension creates the ground truth, while the Squid proxy captures the network traffic on which the same proposed policies will be applied. The test was repeated 5 times.

![Fig. 29. Comparison of the proposed policies with Chrome as testing browser](image)

The average accuracy results from the previous tests are shown in Figure 29. These results validate that the same conclusions hold for both browsers. Comparing with Figure 14, the exact same patterns are visible and similar scores are achieved. More specifically, the Repetition policy already boosts the performance of the Previous policy, while the Next and URL policies offer further improvement. Finally, when the Content policy is combined with all three time-based policies, by far the best results are reached. In conclusion, the above results are a firm indicator that the proposed policies can be safely applied in any case of captured HTTP traffic regardless of the user agent that generated the traffic.

Privacy consideration

The purpose of the proposed policies is to identify user actions and therefore anyone with access to proxy logs could potentially be able to match users with user actions. From that, a user's web history can easily be inferred which constitutes a privacy breach. In order to avoid this, the proxy administrator and everyone else involved in the proxy log handling should take special care in hiding sensitive private data before applying the proposed polices. Such data is the exact IP address of users which can directly identify
Chapter 7: Discussion

a user and the exact full URLs because they often contain user specific information. Since for this thesis only auto generated traces were used, the identified user actions do not belong to any particular user and thus no personal information could be revealed.
8. Conclusion and Future Work

The main goal of this project was to propose several policies for user action identification by only inspecting proxy logs. Proxy logs are in general easier to obtain than server or client logs, thus offering more test data for the browsing behavior research, as well as new insights for the network traffic and web page performance research. The proposed policies are able to process a large amount of network traffic from a proxy log and to a certain degree identify the small number of actual user actions.

The various policies offer different identification rates and by comparing the user actions to the actions suggested by the policies an F1 score of more than 0.80 is attained when a combination of the policies is used. Other proxy solutions have shown to achieve an F1 score of around 0.90 [17], but they are limited in user actions performed in specific websites, while the policies presented here recognize actions on any website. Therefore the presented results open the way for more generic user action identification from proxy logs.

In this effort to achieve user action identification from proxy logs, the vast complexity of modern websites was revealed. The numerous and varying objects within a web page as well as the broad range of techniques for loading a web page render a ubiquitous solution to the action identification problem impossible. This project shed some light on the various page loading techniques and on the modern ways of browsing the Internet. For instance, it was shown that a plethora of websites often generate network traffic related to advertisement and/or tracking services right before they are unloaded in order for a new page to be loaded.

In the direction of evaluating the proposed policies, the ground truth for user actions was required. The ground truth contains the actual user actions that the user performed while navigating the Web with the help of a web browser. In this project the ground truth was produced by a browser extension and the structure, as well as the capabilities of this browser extension are presented. Therefore, in case more policies are to be evaluated or richer datasets are needed, the browser extension presented here can easily be used to produce more ground truth data.

While this project helps the research community take a big step towards user actions identification from proxy logs, there is still room for improvement. One major direction of future work is adaptive parameter selection. This constitutes of alternating the policies’ parameters while analyzing a proxy log in order to maximize the identification accuracy in the form of precision, recall or a combination of them such as the F1 score.

The fine tuning of parameters could be done according to various factors. For example, as seen in related articles, although it is difficult to find a general behavior model for all the different web sites, it is easier to build behavior models for categories of web sites. In this direction, the tuning parameters of the various proposed policies could be adjusted depending on the category of web sites under scope.

Furthermore, the thresholds of the Previous and Next policy could be adjusted depending on the proximity of the proxy server and the web site’s server. Web sites with short round trip time can have a lower Previous-window because all the embedded objects would have arrived before this threshold. For example, if a user browses through a set of web sites that are fully loaded in 500 ms and assuming that a user needs at least 1 second between two consecutive user actions, then having a Previous-window equal to 500 ms would result in finding correctly all the users actions without any false positives. Web sites with longer round trip time should have a greater Previous-window to avoid an excessive number of false positives.

Regarding the Next policy, web sites with short round trip time should have a very low Next-window or maybe the Next policy should not be applied to them at all. That is because, the loading of the objects of these web sites is done so quickly that it seems like it has happened in one burst. On the other hand, a longer round trip time would allow for more false positives to be eliminated by the Next policy.

In a long term scenario a user may visit the same web site multiple times, so the Repetition policy needs a threshold to allow it to forget URLs. If a URL path has been requested before the forgetting threshold, then it should not be filtered out by the Repetition policy.
Finally, another aspect that could possibly be of interest is the grouping capabilities of the proposed policies. Although, the focus of this project was on suggesting user actions, matching HTTP traffic to user actions could be even more useful. For example, a user requests a web page and before the whole web page is fully loaded, the user requests a new web page. After a user requests the second page, objects from the first page may still arrive. These objects will never be displayed to the user; hence wasting valuable network resources. Furthermore, if the user is not charged with a flat-rate, but a usage-based rate, which is the common practice nowadays for 3G/4G networks, these unnecessary objects will result in unnecessary charges. However, if a proxy was able to match network traffic to user actions by just examining the network traffic that goes through it, then it could stop the transmission of the unnecessary objects, thus saving network resources and possible network charges.
References


References


På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Uphovsmannens ideella rätt innefattar rätt att bli nämnad som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: http://www.ep.liu.se/