Design and development of a web traffic analyser

Salim M Ali1, Mustafa A Neamah2
1,2 AL-Nahrain University, College of Information Engineering, Baghdad, Iraq
mustafa.kadhim@coie-nahrain.edu.iq
salim.albayati@yahoo.com

Abstract. This paper focuses on web traffic analysis; we explore the complexity of modern web pages by tracing network packets for the types of objects requested when loading a web page. Furthermore, web pages are not analogous to each other, their design and implementation relies on the purpose of the website. Similarly, web pages that are designed for a specific area, say e-commerce, would have the same development process, which then reflects on the code and objects used to develop such websites. This will lead us to a speculation that web surfers who have similar intention of browsing, they will have similar web traffic in terms of number of different objects and the total size of data transferred through the network. Conversely, inspecting the type, size and number of objects within the browsing session will show us an approximate of what the users’ intention to browse or like to browse next.

1. Introduction
Web browsing have tremendously changed since it was first introduced, modern web pages are not only collections of simple passive objects like images, html codes, scripts, and etc. but they are also breathing portals which interact with its users and other background applications all together to present an active top level model, which sometimes exceed the aptitude of high level applications. This new era of web interface affects the overall internet bandwidth, traffic, and number of sessions.

To summarize our implementation for achieving the above concept, we log user’s browsing sessions and depending on what type of objects were requested for during the session we try to give a general idea of what the user is mostly interested in or likes to browse for. The results obtained from our tool, which are range of statistics, summarizes how many objects were requested for, how many of these requested objects were loaded onto the webpage, how many were already cached, how many images, etc.

The resulted data will help us to produce a learning curve to predict a useful criterion of what the user will browse next, or in general how the traffic statistics going to be. Consequently, the results can help service providers categorize their users and offer them the best type of service enhancing both performance and utilization of available resources. This data can also tell us what type of web pages do users like to visit more often, providing an insight to what type of websites attract more users and from which category.

2. Background
We have noticed that most sniffing tools are designed to sniff and filter packets depending on the packet headers up to transport layer of OSI model. These tools are incompetent of helping us to achieve our project goals, in other words, tracing and categorizing objects from web browsing sessions which uses HTTP protocol. So, we have to create our own tool to be tuned for such purpose. Open source capturing libraries, like Pcap, would be an excellent choice as a starting tool. These kinds of tools will serve us with a base for packet capturing, then we will add more routines to expand the tool’s capability for further packet payload processing [1].

The tool is designed to sniff over a network while browsing the internet and give a summary of what type of web pages were visited. Internet browsing uses TCP over port 80 to request media and other
types of objects that are loaded on a webpage. The initial requests that are made for different objects
and the webpage itself are HTTP GET and POST requests. These HTTP packets are like any other
protocol packets containing a header section and data section. We parse the header section of these
packets to get information about the different MIME types being requested for. But the request packets
often do not contain the complete information about the object. The responses to these GET or POST
requests to the server are also HTTP packets and contain much more information about the object
including type and size, which we primarily use to get information about the object [2].

The responses from the server are not always positive, since the object might no longer be available
or there might not be an object entity to send back, etc. There are different status codes/ messages that
can be received from the server corresponding to the request.

3. Motivations and objective
Internet users are increasing each second, resulting of tremendous amount of data traffic exchange across
the planet. This is not a new fact, but with the entrance of social networks in the beginning of 21st
century has increased the time which internet users spend on browsing and interacting with various types
of web apps periodically and continuously. The other pole which significantly contributes to the big data
exchange world is the recent generations of smart phones and tablets, which some would say that are
not as effective as computers but actually recent statistics shows the amount of web data traffic used by
smartphones exceeds that of its computer companion. Sharing and exchanging media, text, and
application objects became a thriving aspect of users’ interest. Therefore, based of the above reasons we
think that counting web objects and analysing it would be a beneficial tool to aid in traffic usage
minimization and management. The other motivation of choosing such project is the ability to expand
the topic to encompass other network applications that lie above TCP stack protocol. So, the new
implementations can rely on the previous work.

The objective of this tool is to log browsing session and then try to use the data recorded to try to
categorize users, based on the type of web pages they like to view. This categorization can be used by
service providers, etc. to provide the appropriate service to customers enhancing utilization of available
resources.

4. Methodology
The basic functionality of the tool that we built is to sniff TCP packets over a given interface. The
program is setup to accept 2 command line arguments, the interface to sniff on and number of packets
to sniff. If these are not specified the program sniffs on Ethernet 0 and keeps sniffing until an interrupt
is received. The packets sniffed are then segregated to extract different header information. Ethernet
header, IP header and TCP header are extracted and a pointer to the rest of the payload is returned [3].

This payload is then sniffed to obtain the HTTP packet information which contain detail about the
requests made to the server for objects on a web page and the responses received from the server
corresponding to the request. The responses are parsed for more information about the object like type,
size, etc. The information from these packets is saved to two different files, one for requests and one for
responses, for analysis later on. Once the program execution is terminated, it gives out the number of
requests, number of objects loaded, number of objects cached and no of object that were not found or
had no content. These values are also written to a file for reference later on.

The information written to the files is analysed to give statistical data about the user’s browsing
session. This analysis is done using another program which accepts the number of objects the user wants
to analyse from the session data logged. Information obtained contains the number of media objects,
text objects, cached objects and other non-object responses returned. It also parses the file for distinct
IP addresses giving an idea of how many different sources were the objects fetched from. This analysis
is just an example of how this data can be used for further processing. We could filter and analyse the
file for more complicated measurements [4].

5. Model design
The model was designed based on the flowchart below shown in figure 1, basically the user will start
the traffic by surfing the web and it will generate web traffic, then the model will capture the traffic and
it will process it and classify it using different criteria that are based on the user traffic classes. The model will also tell how many images or text objects are there also the size of each object will be shown.

Figure 1. Model Design flowchart

6. Analysis and the mathematical model
In this section a mathematical calculation has been done. To capture a packet time the end and start time capture were subtracted and calculated in millisecond. Total packets captured were divided with the total number of packets to find the Average packet size. We calculated the Data transfer rate (kbps) by dividing the total captured packet length in Kbs with time. At the end we calculated the total data transfer by the following equation [5].

\[
\sum_{t=0}^{n} C_{bytes}
\]  

(1)

Where, \( C \) bytes is the captured packet length and \( n \) is the number of captured packets Traffic capturing on an interface is accomplished using the pcap library, which is a portable C/C++ library for network traffic capture. We decided to use this as it is very simplistic approach and has ample documentation available. There are many good tutorials explaining how to use this library effectively as well. The library has many predefined routines which greatly ease the process of capturing network packets.

The program is setup to accept at most 3 arguments. The user can specify an interface to sniff on and the number of packets to sniff. The default values for both these arguments are used in case the user does not specify either or one of them. The default value for number of packets is -1, meaning that the program will keep capturing packets until an interrupt is received and for the interface is Ethernet 0.

The process of capturing starts with initializing a device to sniff on. If the user has provided an interface, the program tries to initialize sniffing on that port if not it gives an error. Once a device has been initialized, we tell pcap to open this to sniffing network traffic. Once this has been achieved successfully, a filter is initialized so that we only capture packets that needed for analysing browsing sessions. For our implementation, this initialized filter is TCP since all HTTP/HTTPS GET and POST requests are made using TCP. After this, the program is ready to sniff packets off the network.

The next step is to segregate the packets sniffed and extract the relevant information. Since a packet contains Ethernet, IP and TCP headers, we first have to create structures for all these different types of headers. The pcap routine, pcap_loop(), is used to continuously sniff packets over the network. The number of packets to be sniffed is either provided by the user or is infinite. This routine, upon receiving a packet, passes a pointer to the packet to the packet handler function. The packet handler function starts matching the bytes pointed to by the pointer with the structures starting from the Ethernet header structure, the IP header structure and then the TCP header structure. A pointer to the payload portion of the TCP packet is passed to another function which parses through the payload to look at the objects [6].
The packet parsing function checks the first few characters of the payload to identify if the packet is a HTTP/HTTPS request or reply. If the packet is a request the first 300 characters of the packet are saved in a file name HTTPheader.txt. This is done to keep track of all the requests that were made and to check that the reply packets captured correspond to these requests. If the packet is reply then the object type, object size, the port and the sender IP address is written to another tab delimited file named objectype.txt. While parsing replies, we check for status codes as given below:

- 200 - OK: the object was found and request was fulfilled.
- 204 - No content: there was on object entity to send back.
- 304 - Not modified: object has the latest copy already cached.
- 404 - Not found: no objects were found corresponding to the URI in the request.

A count for all these types of objects is maintained and displayed out once the program is terminated. These counts are also written to a file named Traffic_out.txt. The following figure shows a snapshot of the Traffic_out.txt.

![Figure 2. A snapshot of objects statistics of 45 minutes browsing.](image)

To test this implementation, we also created a few test pages on a server. Some of these pages had huge images, some had videos, some had multiple small images, etc. This was necessary to analyse the behaviour of the program. It is extremely important that the number of requests be equal to the sum of all other counts. In some cases, thus pages like these would be perfect to test our approach. For such simple pages, the results were extremely satisfactory and the program’s behaviour was as expected. Hence, we decided to test on it on real web pages.

It was essential to look for simple web pages as there would be no loses and it would be good approach to slowly test for larger web pages. This was we would know exactly where the program misbehaves. Results for simple web page were extremely satisfactory. But as we moved to more complex web pages with a large number of objects, we noticed that the counts were not always equal. Every time the counts were not equal, the number of requests were higher by about 1 or 2 counts on average. The only explanation for this was that some objects are being requested for more than once due to loss of earlier request or reply. We traced through the HTTPheader.txt to look for duplicate requests, in most cases this was true. But it is hard to say with surety that the same is true for all cases since there are too many requests to look at in case of complex webpages.

After our web tool collects the required information about each object such as: type (e.g. image/JPEG), size, TCP port used to download this object, and the web server IP address that this object resides, the overall data will be stored into a single file “objectype.txt”. The file can be seen as number of rows, each row has one object with its information. The basic structure of the file enables the analyser to quickly parse the data file and compute the useful statistics. The analyser basically will consolidate objects originated from the same web server, which have the same IP address. Therefore, each distinct IP address will have its own statistics such as: Number of image contents, text objects, and others. Hence, the analyser can be easily expanded to be much specific about the object type, for example, image content might be JPEG, GIF, or PNG.

The outcome of the analyser would be another file, which is mentioned earlier, this file should contain all the statistics related to the type of objects in terms of number and size and source IP address it come from. The following snapshot shows a small portion of the analyser output,
In short, we run the analyser on objecttype.txt to transform it into useful statistics. This program parses through the file for different object types and distinct sources the objects were fetched from. It counts the number of image objects, text or script objects and other objects loaded on the page and then write the output to a file, out.txt.

Table 1 shows some statistics of different browsing methods, like shopping, kids, and news websites. Hence, top 10 popular websites were used for each category.

| Class     | No. of images object | No. of text objects | Total image objects size (Bytes) | Total text objects size (Bytes) | Total No. of IP addresses accessed | Image to text object ratio | Image to text No object size ratio |
|-----------|-----------------------|---------------------|---------------------------------|---------------------------------|-------------------------------------|---------------------------|-------------------------------|
| Kids (10) | 867                   | 403                 | 9393715                         | 2304027                         | 156                                 | 2.15                       | 4.077                         |
| Shopping (10) | 1812                | 538                 | 18255859                        | 3060362                         | 279                                 | 3.37                       | 5.96                          |
| News (10) | 1756                  | 658                 | 18141979                        | 3694925                         | 352                                 | 2.66                       | 4.91                          |

7. Conclusions and future work

The final implementation of the tool parses all the packet payloads in real time and provides a simple statistic to give a general idea about what type of websites does a user browse for. The tool has been tested for multiple browsing sessions, containing both simple and complex web pages. The results obtained are highly satisfactory and provide a very simple statistic for each session. This statistic is returned by the analyser tool used after the packets have been parsed and the statistic is a very simple example of what could be used for further processing. It also gives you all the distinct IP addresses the objects were fetched from. The tool can be expanded to identify other different types of objects like video files, audio files, css files, js files, etc. giving a broader picture of the browsing session. Data can be sorted with respect to IP addresses to find out what type of service does a web site or source provide.

Our web tool can be a really handy for corporate service providers and network managers to analyse traffic. The analysis can be used by network managers to see what type of users use a specific network and then provide the appropriate architecture. Similar analysis can be used by service providers to allocate and provide the right type service to its customers. Future scope of this tool is vast. It can be expanded to provide summaries of areas or modified to give a much more informative and categorized diagnostic. A drawback of the tool is that it currently cannot account for cached objects, which can be accounted for by parsing the corresponding GET or POST request from the HTTPHeader.txt file. Also, the tool doesn’t tell us about an application that might be running on the web page like flash player. A database could have been used in place of text files, reducing the amount of processing time and calculating different types of statistics will be much easier. These drawbacks could be really good next steps to expand this project.

References

[1] N. K. Hoong, P. K. Hoong, I. K. T. Tan, N. Muthuvelu, and L. C. Seng, “Impact of utilizing forecasted network traffic for data transfers,” Int. Conf. Adv. Commun. Technol. ICACT, pp. 1199–1204, 2011.

[2] M. Kumar and M. Meenu, “Analysis of visitor’s behavior from web log using web log expert
tool,” Proc. Int. Conf. Electron. Commun. Aerosp. Technol. ICECA 2017, vol. 2017-Janua, pp. 296–301, 2017, doi: 10.1109/ICECA.2017.8212820.

[3] N. Shlayan, P. Kachroo, and S. Wadoo, “Bayesian safety analyzer using multiple data sources of accidents,” IEEE Conf. Intell. Transp. Syst. Proceedings, ITSC, pp. 151–156, 2011, doi: 10.1109/ITSC.2011.6083122.

[4] T. Prakash, M. Kakkar, and K. Patel, “Geo-identification of web users through logs using ELK stack,” Proc. 2016 6th Int. Conf. - Cloud Syst. Big Data Eng. Conflu. 2016, pp. 606–610, 2016, doi: 10.1109/CONFLUENCE.2016.7508191.

[5] M. Rahman, Z. I. A. Khalib, and R. B. Ahmad, “A portable network traffic analyzer,” 2008 Int. Conf. Electron. Des. ICED 2008, pp. 1–6, 2008, doi: 10.1109/ICED.2008.4786787.

[6] A. Tiwari and S. Chaturvedi, “Optimized technique for ranking webpage on search engine optimization,” Proc. - 2nd Int. Conf. Micro-Electronics Telecommun. Eng. ICMETE 2018, no. 2016, pp. 107–110, 2018, doi: 10.1109/ICMETE.2018.00034.