Simplifying the Web: Optimized Network Architecture and User Interface for improved Internet access in Developing Regions

Gabriel Garcia Leyva

Computer Science Capstone Project

Advisors: Yasir Zaki, Moumena Chaqfeh

Abstract

The Internet has developed since its initial consumer use in the 1990's to boast more than 4 billion users. To support this exponential growth the developed world has spent countless resources on deploying high-speed networks that can compensate for the Internet's increase in complexity. Modern web pages are riddled with inefficiency, which makes them heavy and slow in places that lack the proper infrastructure. This leaves developing regions and remote areas with a sub-par and many times cost prohibitive Internet experience, as most users in these regions access the Internet through mobile data networks. We aim to provide an optimized Network Architecture and User Interface in developing regions, so they too can take advantage of the wonders of the Internet. Our research focuses on deploying Man-In-The-Middle (MITM) Proxies on Distributed Edge Caches that are close to the users and store cached pages, which are optimized through our JavaScript simplification software. We also provide a light-weight User Interface that allows users to search our database of cached pages through relevant keywords, thanks to our word-ranking algorithm. Our Results showed that our MITM Distributed Edge Caches provided significant improvements in Page Load Time (PLT), Page Size, and Number of Requests.

Introduction

Throughout the years the Internet has become exponentially more complex. Web pages that used to be written in HTML, with a little JavaScript logic and some CSS resources, are now written in React, Angular, or Node, and riddled with inefficiencies. Many resources are now imported from external libraries. Such level of complexity leads to a large number of inefficiencies and unnecessary resources being requested, which leads to an increased Page Load Time (PLT), Page Size, and Number of Requests. The developed world has compensated this issue by building large network infrastructures that allow for high-speed Internet access. However, developing regions and remote areas have not been able to deploy such expensive infrastructure, and are left with a sub-par experience. Furthermore, The International Telecommunication Union (ITU) reported that mobile-broadband subscriptions reached 2.3 billion by the end of 2014, with 55% of them in developing countries[1], as most users in the developing world only have mobile-broadband subscriptions. Given this scenario accessing the Internet becomes cost-prohibitive as sites they access unnecessarily consume much of their data on unused resources.

However, this sub-par experience is not only caused by inefficient web pages. The Page Load Time (PLT) of the web pages that users in developing regions access is also hampered by the large Round Trip Times (RTT) that the requests require since most web pages' servers are located in North America and Europe. The critical limitations characteristic of network infrastructures in developing regions do not help alleviate this. First, they are largely unreliable and outdated. Second, Internet Service Providers (ISPs) in these regions have spent little to no resources on reducing RTT by placing servers closer to their users, which increases the expected Page Load Time (PLT) and becomes a bottleneck in the improvement of networks in these areas. Therefore, a combination of large Page Load Times, Page Sizes, Number of Requests, and unreliable networks limits at least and blocks at most the access of users in developing countries to what most in the developed world consider a basic necessity.

Specific Aims

To address the aforementioned issues of slow and heavy web pages, we aim to develop an optimized network architecture in order to reduce Page Load Time (PLT), Page Size and Number of Requests of all requested pages. We aim to achieve these results through a combination of better network design and JavaScript simplification, by caching simplified pages.

In order to allow users to interact with our optimized network architecture, we aim to develop a User Interface, consisting of a search engine that will allow the users to search for our cached simplified pages. We also aim to make this search engine as "smart" as possible, by adding a word-ranking algorithm which we expect to lead to more accurate search results. Finally, we expect this technology to be implemented in developing regions and remote areas, which leads us to develop a light-weight and simple software.

Background

We have analyzed state-of-the-art technologies relevant to the implementation of our network architecture. Our solution either builds on top of such technologies or fares better in terms of Page Load Time (PLT), Page Size, and Number of Requests. We have seen how using the SpeedReader[2] reader mode improves the load time of HTML elements by performing a tree translation that removes useless page elements. Along those lines, we studied Polaris[3], which uses fine-grained dependency graphs to dynamically determine which objects to load, and when. Concerning limiting Page Size we delved into Flywheel[4], an HTTP proxy service that extends the life of mobile data plans by compressing responses in-flight between origin servers and client browsers and reduces the size of proxied web pages by 50% for a median user. However, three papers are essential to the successful completion of our project.

Shandian

From Shandian[5] we get the way in which we load the pages and the inspiration to do computations in a proxy server. Shandian improves Page Load Time (PLT) by simplifying the client-side page load process through an architecture that splits the page load process between a proxy server and the client. By performing pre-processing in the proxy server which has more computing power, Shandian largely reduces the inefficiencies of leading to slow Page Load Time (PLT) on the client. Shandian is fast for displaying web pages, ensures that users are able to continue interacting with the page, and is compatible with caching, CDNs, and security features that enforce same-origin policies.

XCache

From the xCache paper[6], we get the core of the architecture that we use in our network design. The xCache architecture consists of a set of Edge Caches (ECs) in close proximity to the users. ECs are centrally monitored and managed by a Cloud Controller (CC). The CC collects web page requests and performance information from the ECs to determine the best set of web pages to prefetch and store at each EC. The Cloud Controller is in charge of simplifying the web pages requested by the ECs and sending the cached versions back to the ECs for easy and fast access by the end-users.

Understanding Mobile Phones Characteristics In Developing Countries

We also considered important to understand the characteristics of the devices, users in developing regions access the Internet with. This paper uses 2014 data of about 0.5 million users in Pakistan[1]. Although the data is somehow outdated, there are important takeaways that we can extrapolate to the present. First, lack of enough memory (RAM) and CPU power is a very large issue in mobile devices overall, but especially in developing regions. That is why we have decided to take that burden out of the user's device and put it in our Cloud Controller (CC). Second, even if network conditions improve in developing regions, there might be many bottlenecks at the device level that we should take into when simplifying we pages.

The above literature, although relevant, does not match our solution's performance, which is one of the key reasons why we decided to conduct this research. First, all of the above technologies address part of the problem, rather than having a holistic approach to the issue of Internet access in developing regions, therefore, alleviating certain pain points but not all of them. There are two major issues that lead our solution to perform better than the existing literature. First, when addressing network conditions, they lack a User Interface that allows users to easily interact with their solutions. Second, they fail to address the main issue that undermines the modern Internet: JavaScript inefficiencies, which result in unnecessary URL requests and imported resources.

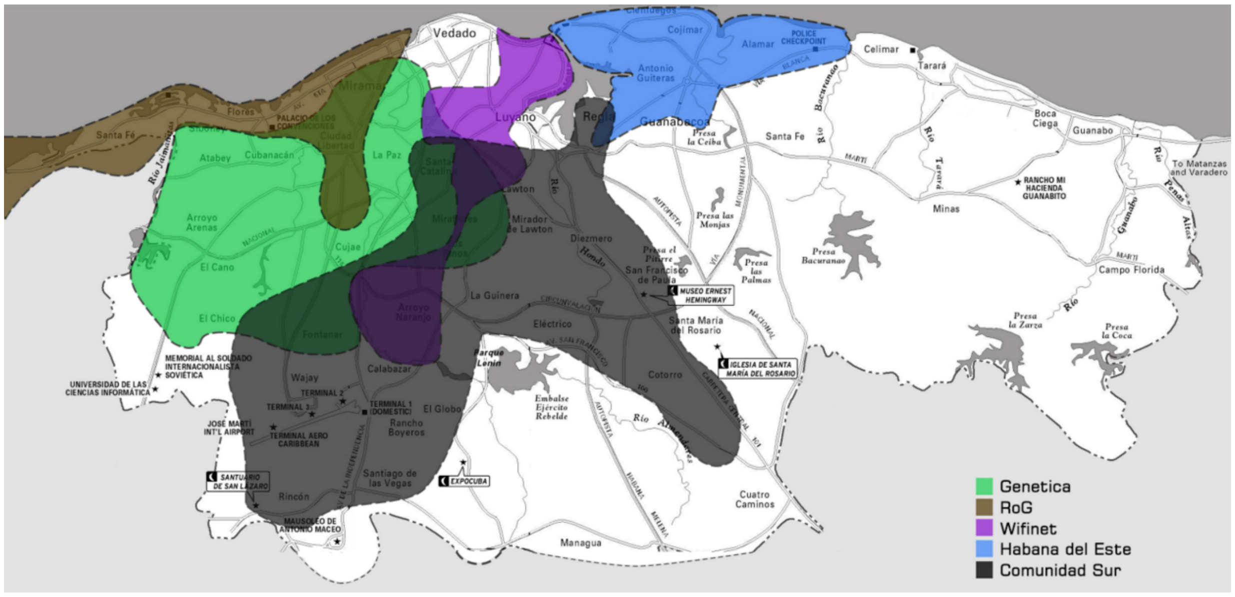
Goals And Potential Impact

The goal of this project is to improve the overall experience that users enjoy when accessing the Internet while on a limited and/or unreliable networks. The improvements we aim to offer can become a huge improvement in comparisons with the options that users in the developing regions currently have. In addition to the aforementioned technical aspects, our research project can also have a direct impact on the economy of our users. Since we reduce the Page Load Time and Page Size of requested web pages, we save the user both time and data, therefore, helping our users regardless of the way the Internet usage is measured and charged in their countries/areas. Our hope is that this project positively impacts users across developing regions and remote areas elsewhere who still struggle with connectivity and/or are priced out of the current Internet offerings. More specifically, as part of our research, we expected to deploy our solution in Havana, Cuba in partnership with the University of Havana. However, given the new environment caused by the COVID-19 pandemic, this was not possible. This notwithstanding, we describe below our implementation plan in the hopes of future deployment:

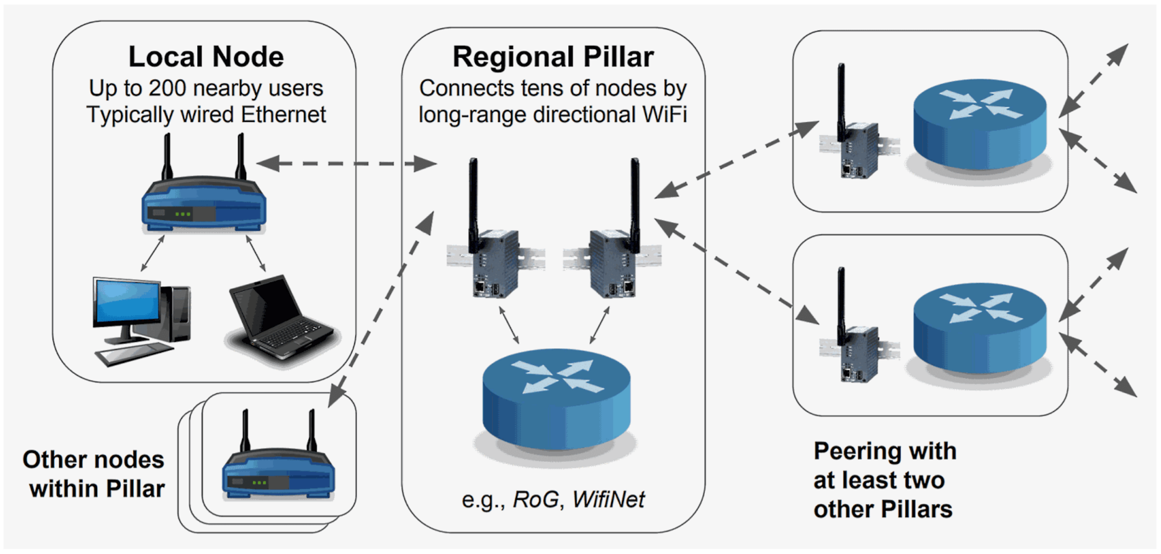
In Havana, Cuba we planned to serve 2,000 members of a community that currently boasts more than 5000 users, who have no access to the Internet. We decided to only serve 2,000 users, as those are the ones closest to our University of Havana hub. These users are part of what is colloquially known as the "SNET" (Street Network), which is a completely offline Metropolitan Area Network (MAN). If successful we would reduce the cost of accessing the Internet for this pool of users and potentially for the whole country, assuming that the only Internet Service Provider (ISP) in Cuba, ETECSA, a government monopoly, picks up on our idea as a way to improve Internet access.

"SNET": Cuba's DIY Internet Network

The SNET is a very complex Metropolitan Area Network (MAN), established in the late 90's. It has since seen exponential growth, allowing it to cover large portions of the city. The network has five main regions in Havana, which in many instances overlap with each other since the only determinant of a region is the Regional Pillar users are connected to. These regions have at least 1000 users each. They are called: Genetica (Western Havana), RoG (North-Western Havana), Wifinet (Central Havana), Habana del Este (North-Eastern Havana), and Comunidad Sur (Southern Havana).



The Network architecture of the SNET is worth understanding due to its unusual design. The main driver of this unique architecture is the DIY (Do It Yourself) idea that drives the network and the way new users join. Up to 200 users are connected locally through Ethernet cables of up to 100m, which go through streets and street electricity lines from one block to the other. All of these users are connected to a Local Node. This Local Node is connected to one of the five Regional Pillars through long-range directional WiFi, along with tens of other Local Nodes. Each of the Regional Pillars is Peering with at least two other Regional Pillars, which allows for the whole city to be interconnected without the use of the Internet.



Each Regional Pillar has at least an administrator. These administrators, along with skilled members of the community have developed software for the network, including their own Facebook-like social media, hacked versions of popular multi-player games such as ”Call of Duty” and have created a dedicated browser. Therefore, integrating our User Interface as an extension into the network would be a seamless process.

Methodology

We will now delve into the two elements of our project: the Network Architecture and the User Interface, delving into the details of our implementation.

Network Architecture

In our network implementation, we have heavily drawn on the xCache architecture, although we have added certain modifications in order to make it more efficient. The XCache architecture is described in Figure 5 below. From xCache we use Edge Caches and Cloud Controller elements. However, we add several modifications necessary for our network architecture to be relevant.

To web clients, an Edge Cache functions as an ordinary proxy or cache that receives users HTTP requests and in the case of a cache hit, serves the objects directly. In the case of a cache miss, the request is relayed to the Internet. ECs differ from conventional caches in two ways: First, ECs are managed by a Cloud Controller and are periodically updated with web objects. Second, ECs gather access logs and periodically send the aggregated statistics to the Cloud Controller. The main goal of the Cloud Controller is to optimize the content of each EC by maximizing the ECs hit rates while minimizing the bandwidth used to update them.[6]

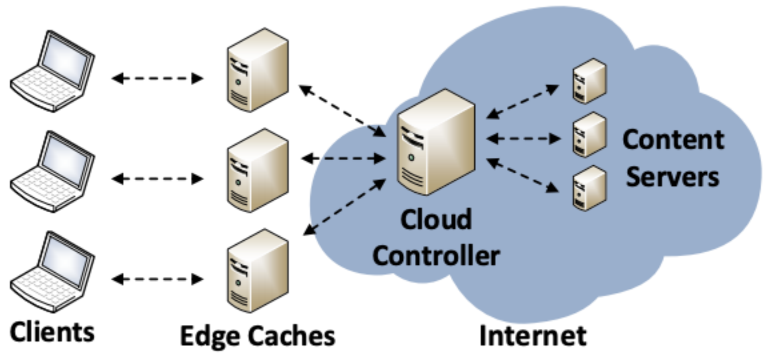


Figure 5: xCache architecture

The modifications we have added to the xCache architecture are as follows. First, the Edge Caches behave in a Peer to Peer (P2P) manner, on what we initially called Distributed Edge Caching. However, the term has been coined in other papers such as "Distributed edge caching scheme considering the trade-off between the diversity and redundancy of cached content"[7] with a similar meaning but without the P2P aspect to it. Therefore, we have decided to call it P2P Distributed Edge Caching. In P2P Distributed Edge Caching each Edge Cache announces the pages they have cached along with a time-stamp and version number, which other Edge Caches can request as needed. There are two reasons for this decision. First, since different Edge Caches will serve different sets of users they should each have a different set of pages cached. However, users served by one of the Edge Caches might ask for a page that is already cached in one of the other Edge Caches. By using a P2P architecture at the Edge Cache level we avoid a whole round trip to the Cloud Controller and/or Internet, thus saving time and resources. Second, not every Edge Cache will have the latest version of a certain web-page, which would force the Cloud Controller to update all Edge Caches when updating a web page. However, by allowing Edge Caches to share information among themselves the Cloud Controller only needs to update one Edge Cache and then that Edge Cache will share that update as needed with the rest.

As a proxy, every Edge Cache will intercept the requests users make to the Internet. We have chosen to implement this interception through a Man-In-The-Middle (MITM) proxy. MITM proxy is an open-source library that allows to intercept network requests and responses. We interact with the MITM proxy through a Python API, which has several modules: mitmproxy(), mitmweb(), and mitmdump(). The first one, mitmproxy(), is a terminal interface that keeps track of the GET and POST requests and allows developers to examine each request in-depth. The second one, mitmweb(), is the same as a mitmproxy() but displayed in a web browser environment. The third and final one, mitmdump(), is the one we use, and it is a simpler version of mitmproxy(), simply writing the GET and POST requests directly into the output window in terminal. This added simplicity allows us to write the traffic into a file and have more control over the MITM proxy implementation. Moreover, this module allows us to add a certain level of personalization to the behavior of MITM proxy. For this purpose, we developed a python script that controls the behavior of the MITM proxy once the proxy intercepts GET and POST requests, which is one of the options of the mitmdump() modules. The python script then interacts with our database set up, which we have created in order to offer a cache framework for the program. For this database, we have chosen to use a MySQL database, storing request URLs as primary key and the relative path of the response files as the other two fields, making the database only store metadata. We have made the decision of only storing metadata in the database in order to have the flexibility of changing the actual response files. The response files are saved as two files, a header file (.h), storing the HTTP response headers, and a content file (.c), storing the content. However, we do not store the response files directly from the Internet. The response files are simplified by the Cloud Controller using the JSLearner simplification software developed by our teammates Jacinta and Manesha, which combines script labeling and URL blocking. That simplified versions of web pages are the ones we store in the database.

It is also important the way we implemented the database, as several technical limitations lead us to these decisions. Given that the primary key of any SQL database needs to indexed in order to offer an access time of O(1), there is a length limitation of 256 characters for the index field. Request URLs can be longer than 256 characters, which is the reason why we decided to hash the request URLs, store the hashed URL requests in the database, and unhash them once we need to check them. For GET requests, we unhash the request URL in order to check whether the request URL has been cached in our database. If it has, we do not allow the request to make it past the proxy, instead, we create a GET response at the proxy level and feed it the simplified cached version of the response files. If it has not been cached, we allow the GET request to go through to the Internet as it normally would. Once the GET response is sent back after some time, we store the response files in our server and register their relative path in the database associated with their hashed request URL.

User Interface

In order to allow our users to interact with our cached pages, we came to the conclusion that we needed to develop a User Interface that was both simple to use and lightweight. In order to address the first requirement, we decided to go with a search engine implementation in the front end. We fulfilled the second requirement by coding a simple search engine landing page in HTML, basic CSS styling for the search bar, button, item lists, and thumbnails, as well as a background image. Furthermore, our User Interface only uses one JavaScript function, which is used to generate the GET request for the search term and to handle the POST response, by dynamically inserting the thumbnails and URLs into list (li) HTML elements. It is important to note that our User Interface could be deployed as a browser extension for Google Chrome and Firefox, as the most used browsers globally. However, in the case of Havana, Cuba we would have not implemented the User Interface as a Google Chrome or Firefox extension, as the Metropolitan Area Network (MAN) we were targeting developed their own light-weight browser.

Our decision to search engine homepage stems from the standard user behavior of searching for anything online and expecting to see the Google search results page when searching from the Google homepage or from the browser search bar. Given the origin of our decision, we established the need to mimic the standard searching behavior as much as possible. We mainly focused on making our search engine "smart". Our primary goal was to deliver relevant results to any search terms our users might type into our search bar. In order to satisfy these requirements, we decided to develop a key-ranking algorithm. We used the Natural Language Toolkit (nltk) to rank the 5 most frequent words in every website, removing any stop words like "the" or "and", which would not lead to relevant search results. Once deployed the Cloud Controller will run this script using a headless Selenium script on every page it caches and simplifies. The resulting keywords along with the domain name (for specific hits) will be stored in a MySQL database table in the Edge Cache. This MySQL database will be different from the one used by the MITM proxy, as it will act as the back-end of the User Interface. This database consists of the following fields: requests URL, keywords, and relative path to the thumbnails associated with each URL. In the average scenario the user will search for a keyword and relevant results literal or not will appear as a xlist right below the search bar. The user will then be able to click on whichever link they wish to. That request would be intercepted by our MITM proxy and the user would be served the cached version of the page.

Design Choices

Given the requirements of our project we have had to compromise on certain aspects to meet the large goal of providing an improved and inexpensive Internet experience to developing regions and remote areas. Therefore, we had to make the following design choices:

The first decision we had to make was how to address the first time one or more users access a certain website. Given that we need to fetch, simplify, register keywords, and cache the web page, the request cannot be served instantly. In order to address this issue, we have made the decision to tell the user to come back after a certain time (i.e., 20 minutes). We would then allow the page to be processed by the Cloud Controller and cached by the user's Edge Cache. If the page were to be cached before our estimated time, we would notify the user accordingly

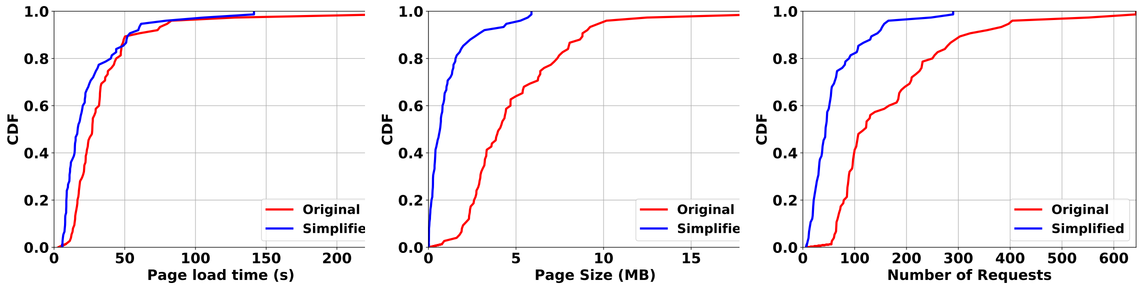
The second decision we were presented with was how to handle pages that require the user to Log In. This is actually the hardest of the problems that we encountered. The issue is two-fold. First, web pages that require logins generate a different front-end for each user, in terms of looks and content. Trying to cache each version of web pages like Facebook or Instagram would be extremely difficult, if not impossible. Second, this type of web page presents a large security risk for network administrators. Users would have to disclose to the network administrators their usernames and passwords. The effort to secure all of that information in addition to the potential legal implications would have overtaken the rest of the project, not allowing us to focus on the actual goal of our research project. Therefore, we have decided not to support pages that require Log In.

Results

Network Evaluations

As previously mentioned we expected to generate on the ground results in Havana, Cuba. However, we were not able to do so due to the global situation caused by the COVID-19 pandemic. As an alternative, we decided to simulate the network conditions we would find in Cuba. However, scholarly literature on the topic is nonexistent. As a consequence, we turned to other developing countries in the hopes to extrapolate network conditions that would approach the average network conditions in developing regions and remote areas. In deciding the network conditions we used to simulate our results, we looked at the state of cellular data connectivity in India in 2014[8], which led us to the following parameters: Downlink Bandwidth of 3,500 kbps; 5% of Downlink Packets Dropped; and a Downlink Delay of 75 ms. These network conditions are inclusive of the improvement in network conditions in developing regions since 2014.

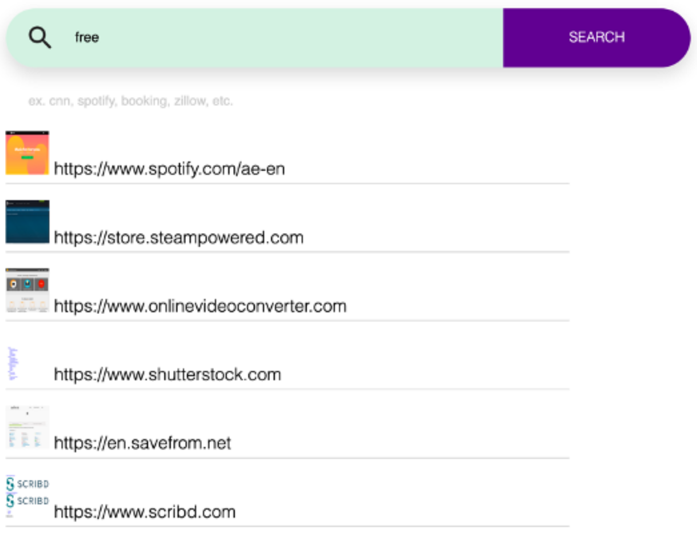
Setting our network conditions to the aforementioned parameters, we evaluated Alexa's Top 75 Popular web pages, both in their original and simplified versions, specifically looking at Page Load Time (PLT), Page Size, and Number of Requests. We obtained this information from the HTTP Archive format, or .har file, which is a JSON-formatted archive file format for logging of a web browser's interaction with a site. We extracted the .har files generated by the browser for each web page. Before we extracted the .har file, we scrolled the web page from bottom to top to make sure it was fully loaded. this process was first performed on the 75 original pages, which were requested directly from the Internet using the Firefox Selenium webdriver. Once completed, we proceeded to use the Firefox Selenium webdriver to request the simplified versions of the web pages through our MITM Proxy Architecture, under the same network conditions.



The leftmost plot represents the CDF results for different Page Load Time (PLT) of Alexa's Top 75 web pages in their Original and Simplified versions. In the case of Simplified pages, we see a Page Load Time (PLT) of about 20% in comparison with the Original versions. It is also important to note that the maximum Page Load Time for the Simplified versions of the web pages is about 140 seconds, in comparison with the 200+ seconds that took the slowest of the Original versions of the web pages. Regarding Page Size, we can observe the CDF result in the center plot. This is the metric we observed improved the most. Our solution managed to reduce the Page Size of Simplified versions to a maximum of about 7 MB, in comparison to a maximum of 18+ MB for the Original versions. Furthermore, it managed a reduction of about 80% in the Page Size of web pages. Finally, we can see the CDF results for the Number of Requests in the rightmost plot. As expected given the PLT and Page Size CDF results, we also saw a significant reduction in the Number of Requests in the Simplified versions of the web pages, with a maximum of about 300 requests per web page. The evaluation showed a reduction of about 50% in the Number of Requests per web page. These results are especially relevant as they confirm the effect we expected our solution would have in benefiting users in developing regions and remote areas.

User Interface

Our User Interface was implemented in the aforementioned manner. It is now ready to be implemented as a browser extension in Google Chrome, Firefox, or any other browser it needs to be used in. We believe our User Interface is very intuitive as it closely resembles all other search engines, including Google.com and Yahoo!. In the below figures, we can see two examples of the search results our User Interface would yield from certain search terms, which showcases our relevant keywords approach.



https://www.spotify.com/ae-en
https://store.steampowered.com
https://www.onlinevideoconverter.com
https://www.shutterstock.com
https://en.savefrom.net
https://www.scribd.com

Figure 1: Homepage with search results for ”free”

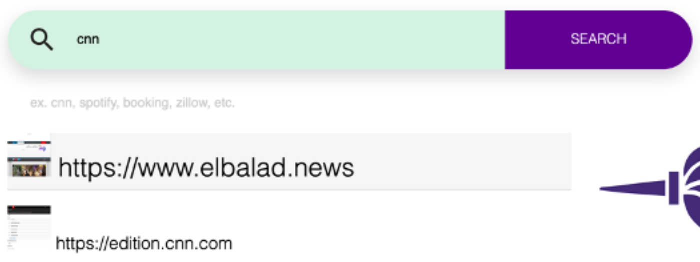


Figure 2: Homepage with search results for ”cnn”

In the above figures, we can see some of the results displayed when the user searches for the word ”free” (Figure 1) and the results displayed when a user searches for the word ”cnn” (Figure 2). The user is then able to click on the desired link and the cached version of the page would be served. If the search yields no results the user can search for a domain and expect to find the cached web page in the search results after some time (i.e, 20 minutes). The user will be notified if the web page is cached in a shorter period of time.

Conclusion

We have found that our solution offers a reduced Page Load Time and Number of Requests, and a greatly reduced Page Size, leading to a significant reduction in Internet Access costs. Our hope is that our solution is implemented in Havana, Cuba, or any other community that might benefit from this solution. We also encourage further research addressing the caching and simplification of web pages that require Log In, such as social media.

References

References Cited

[1] S. Ahmad, A. L. Haamid, Z. A. Qazi, Z. Zhou, T. Benson, and I. A. Qazi, "A view from the other side: Understanding mobile phone characteristics in the developing world," *Proceedings of the 2016 ACM on Internet Measurement Conference - IMC 16*, 2016.

[2] M. Ghasemisharif, P. Snyder, A. Aucinas, and B. Livshits, "Speedreader: Reader mode made fast and private." [Online]. Available: https://arxiv.org/abs/1811.03661

[3] R. Netravali, A. Goyal, J. Mickens, and H. Balakrishnan, "Polaris: Faster page loads using fine-grained dependency tracking," *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. [Online]. Available: https://www.usenix.org/node/194917

[4] V. Agababov, M. Buettner, V. Chudnovsky, M. Cogan, B. Greenstein, S. McDaniel, M. Piatek, C. Scott, M. Welsh, B. Yin, and et al., "Flywheel: Googles data compression proxy for the mobile web," *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. [Online]. Available: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/agababov

[5] X. S. Wang, A. Krishnamurthy, and D. Wetherall, "Speeding up web page loads with shandian," *USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. [Online]. Available: https://www.usenix.org/system/files/conference/nsdi16/nsdi16-paper-wang-xiao-sophia.pdf

[6] A. Raza, Y. Zaki, T. Ptsch, J. Chen, and L. Subramanian, "xcache: Rethinking edge caching for developing regions," *Proceedings of the Ninth International Conference on Information and Communication Technologies and Development - ICTD 17*, 2017.

[7] S. Wang, X. Zhang, K. Yang, L. Wang, and W. Wang, "Distributed edge caching scheme considering the tradeoff between the diversity and redundancy of cached content," pp. 1–5, 2015.

[8] Z. Koradia, G. Mannava, A. Raman, G. Aggarwal, V. Ribeiro, A. Seth, S. Ardon, A. Mahanti, and S. Triukose, "First impressions on the state of cellular data connectivity in india," *Proceedings of the 4th Annual Symposium on Computing for Development - ACM DEV-4 13*, 2013. [Online]. Available: https://www.researchgate.net/publication/262362068_First_impressions_on_the_state_of_cellular_data_connectivity_in_India