EVALUATING THE PERFORMANCE AND OPTIMIZATION OF THE MODERN WEB

Alexander Cossifos

A Capstone Project Submitted to the
University of North Carolina Wilmington in Partial Fulfillment
of the Requirements for the Degree of
Master of Science

Department of Computer Science
and
Congdon School of Supply Chain, Business Analytics & Information Systems

University of North Carolina Wilmington

2024

Approved by

Advisory Committee

Karl Ricanek	Yao Shi	Jeffrey Cummings

Kevin Matthews
Chair

Abstract

Evaluating the Performance and Optimization of the Modern Web. Cossifos, Alexander, 2024. Capstone Paper, University of North Carolina Wilmington.

While hardware has continued to evolve and improve exponentially over time, the software running on it may not include the same improvements. Some could even argue that the performance and efficiency of software has taken a step back. One microcosm of this would be the modern web. This research sets out to investigate different categories of websites by profiling their performance data. Big differences are found in some categories, in terms of performance and efficiency, while in others the differences are negligible. Using the collected data, suggestions of more efficient and performant alternatives to the usual bloated sites most people use are made. Categories like web apps, and comparing them to native versions, are also investigated. A thorough literature review has been completed to substantiate some of the claims being made. Data was collected in Spring 2023 and Spring 2024 to view how things have progressed and if there are any trends. What was found is that some of the bigger players in each category have improved somewhat compared to the last data collection and the underdogs have actually gotten slightly worse. When viewing web apps vs. native apps, it is found that the native apps use more resources but run smoothly. The web apps still have issues with feature parity and while their single process takes less resource, processes related to them make the total more than the native and overall offer worse performance. Whatever is found, it will help to know how to develop more performant and efficient software. It is best to have your software be as close to the hardware of your machine as you can.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

A Brief History

The work computers can do and how well they are able to do that work has generally increased over time. The ENIAC, the first general purpose digital computer, constructed in 1943 finishing in 1946 was made to calculate artillery firing tables, but also could do other numerical problems (IEEE, 2023). This machine was tedious to work with, requiring the use of punch cards and wires to be reconfigured for each program it was to run in addition to the vacuum tubes that proved to be unreliable and had to be replaced frequently (Britannica, 2024; University of Michigan, 2024). Later, the Apollo guidance computer, which helped to land the first humans on the moon, took advantage of the advancements of the time to fit it within the shuttle (Eloon C. Hall, 1972). It was "the control and processing center which processed data and issued discrete output and control pulses to the guidance system and other space craft systems" (Eloon C. Hall, 1972, p. v).

The home computing revolution brought these once institutional and government machines to the average Joe with feature sets to meet those needs. The Apple II, which came out in 1977, was one of the computers at the forefront of that. Though it was relatively affordable at the time compared to what came before, the fact that it contained a text and spreadsheet editor is what really boosted its popularity (Computer History Museum, 2024; National Museum of American History, 2024). It also helped to make video games popular outside the arcade along with the IBM Personal Computer that came out in 1981 (Jennings & Brewster, 1998).

As time moved on, the use of computers by individuals has further thrived and become ubiquitous. The first iPhone started the mobile computing revolution in 2007. It

combined a “mobile phone, a widescreen iPod® with touch controls, and a breakthrough Internet communications device with desktop-class email, web browsing, searching and maps—into one small and lightweight handheld device” (Apple et al., 2007). In 2024, with the ability to take high quality videos, play near full-fledged games, and have near identical creation suites for office tasks, music, as well as scientific applications, smartphones are the main way consumers compute, replacing desktops almost outright and even laptops.

Despite these advancements in hardware and computing capabilities, the evolution of software and development may not have unfolded in a way that benefits users in the best way. A look into existing software products can reveal interesting trends that may provide prescriptions for both developers and end users.

The Problem to Be Explored

As stated earlier, the performance of the software running on the currently available advanced hardware leaves much to be desired. The ability of software to do what people need has been met but some might say it has stagnated in terms of quality. The new features going into today’s software often includes bloat that will not benefit the end user (Shmueli & Ronen, 2017). What is worse is that the updates containing the new features negatively affect how the base functions of the program run. Background processes that are not able to be stopped take up resources and cause major slowdowns despite how powerful these machines are now compared to when functionality peaked over a decade ago for the end user.

One part of this is due to the high-level programming languages and tools used to make modern software. Examples of popular high-level languages today include Java, JavaScript, and Python (Gunnarsson & Herber, 2020). They have made development

easier through simpler syntax, abstraction, and automating processes like memory management via garbage collection, but have we been slowly losing the artform of code optimization because of the freedom granted by ample hardware resources and the ability for code to work without it being near perfected first? The code that went into the early computers, like the one on Apollo 11, did not have any resources to spare in both its computing power and its memory size . The programmers making the software for those computers either had to already be experts at optimization or learn on the job quickly. With optimization not being required by the nature of the environment itself, it is likely many programmers will never truly become as skilled in that aspect as those of the past. To make a parallel involving consumer behavior relating to tech over the past two decades; Streaming and digital distribution has taken over physical media as the main way most people consume entertainment (Statista, 2024). Additionally online shopping sites like Amazon have made it so easy to get anything shipped to their house that it has dealt a serious blow to the physical retail space with many retailers with singular purposes like Blockbuster, Radio Shack, and Toys R Us ceasing to exist partially because of it (Worldwide Business Research, 2019). It may be that just like consumers, developers have chosen convenience first over the actual quality of the product.

Another part of this, sort of on the other side of the spectrum but still related, is the telemetry and data collection done by modern software. With not only ample hardware resources available but high bandwidth internet connections at their disposal software companies started collecting data of users (Dembrow, 2022). It may have started as only technical data, like what version of an operating system is installed, the specs of the computer being run on it, system diagnostics and crash reports that could be very useful to developers; but that all quickly changed once the opportunity to make money

entered the equation. User data, including their name, date of birth, email address, phone number, actual address and much more, including the previously mentioned technical information, is now being sold (Dembrow, 2022). Hidden away in many terms of service and end user license agreements is the clause allowing companies to collect said data and sell it for a great amount of money. And not just once, the same data could be sold many times to different customers wanting to know consumer behavior and market trends. Developers may have done a good job at optimizing software to collect as much data as possible but not in terms of how it affects the overall performance of the computer running the software. What makes this infinitely worse is that with common use cases and workflows, many must use multiple pieces of software that is collecting data all at the same time, including the OS. It can seriously bring a computer back a few generations of hardware or if it is old enough to a standstill.

The current research investigates the modern web as a prime example of these problems at their worst. Google Chrome, the most popular web browser, is already a notorious resource hog (Mighell, 2019; Vailshery, 2024). The investigation includes the websites that most people visit, their available versions, and common software. Modern web browsers like Google Chrome can help to see some of what is truly going on with the use of built in tools. The following research questions will direct this exploration:

1. What exactly is taking up resources, like CPU time and memory, and is it something that can be easily changed?
2. How is software less optimized, especially with regard to the conflict between what developers can do vs what users need?
3. And - based on these results - what recommendations can be made to developers and users?

Chapter 2: Review of Literature

To help answer those research questions, a look at existing literature focusing on software development and performance is in order. First will be a look at web site development and performance, with a look into the desktop vs. mobile versions of web pages as well as landing vs. internal pages. Next, an overview of the current tools used to make software including programming languages and frameworks. Lastly, resources and their different types will be discussed about how they affect performance and energy consumption. Understanding these areas can help set the stage for discussing the exploration of resource allocation and optimization.

Desktop vs. Mobile

When looking at the modern web it can be divided into two separate categories, desktop and mobile. The former having a bigger screen to display more objects and not having to worry about power consumption, then the latter having a much smaller window to view objects as well as a limited battery to power its functions. But there is a third category that gets overlooked because of how it has mostly replaced the desktop but has the power limitation of mobile, such as laptops. Most laptops have screens large enough to display all the content that their desktop equivalents could, so it is fed the same content which is possibly not optimized with regard to its limited power.

A decade ago, it was observed that mobile web resource usage was generally less than desktop. The average desktop web page has over 100 requests in comparison to the average mobile web page having over 60 requests, both steadily increasing over time (Johnson & Seeling, 2013). The more requests that are made, the more energy consumption is increased. In the past, what was mainly responsible for those requests were images which were also responsible to the increase in web page size along with

other components and JavaScript (Johnson & Seeling, 2013) When looking exclusively at mobile, JavaScript was seen as the main contributor to the increase in web page sizes. JavaScript along with CSS were found to contribute most to mobile power consumption as well (Johnson & Seeling, 2013).

Of course, in the past 10 years, things have changed and some of the predictions made are not necessarily accurate. First, Flash, which accounted for a large amount of data per request and was known for being so resource heavy that Apple did not allow web pages to use it on iPhone, has been discontinued (Johnson & Seeling, 2013). Second, the predicted average wait time of 45 seconds for mobile web pages to load has not turned out to be true. Even the low-ball projection of 15 seconds or below of wait times are double of what most actually experience (Johnson & Seeling, 2013).

The web also has a difference in its home, or landing, and internal pages. The home/landing page is usually the first page a user will see so it must make a good impression for them to keep using the site. Though once they are in and go to one of the internal pages, that does not have to be the case anymore. Once that journey is started, usage data can be collected on the path the user took throughout the website, how long they stayed on each page, where the mouse was on the page, etc. Interestingly, landing pages have fewer objects and will load faster typically than internal pages despite them being bigger (Aqeel et al., 2020).

High-Level Languages and Tools

How a web page performs on the surface is only part of the story, there are also the languages, tools, and frameworks used to make and run these sites. Different types are required for different use cases, as such the front end can be entirely different from the back end in terms of how performant and optimized it is. Understanding these options

can help to discuss whether developers are picking the right tools for the job

The tools developers have used to make software have evolved and changed over time. Starting out with ones and zeros, then to assembly, to low-level, and then to high-level programming languages. These high-level languages do a lot of heavy lifting allowing developers to just focus on creating a shippable product. But just because it makes the process of creation easier does not mean the end product is better. Are these high-level languages really the cause here or is it developer error? Are they really what most are using?

In 2020, it was observed that the most popular programming languages were Java, Python, Java Script, and C++ based on statistics from GitHub API (Gunnarsson & Herber, 2020). JavaScript in particular has been prone to developers creating solutions with it that are not particularly efficient and using built in solutions that are not fully suited for what needs to be accomplished (Selakovic & Pradel, 2016). Frameworks for JavaScript, like jQuery and Backbone.js, though having more specialized tools, are slower and use more memory (Ladan, n.d., 2015).

The performance of Python, and other dynamic languages, in tandem with its garbage collection is determined by how its run and other aspects (Ismail & Suh, 2018). C function calls in Python, which are supposed to be built in methods to speed up processing, contribute to a great amount of overhead and if running in Just-In-Time mode aspects such as cache hierarchy and the memory system make a larger difference when it comes to performance and garbage collection (Ismail & Suh, 2018). Though recently certain techniques like learned garbage collection could help to adjust their behavior to different needs (Cen et al., 2020). Others in the past have found that because of modern multicore processors that have the ability to negate the negative performance impact

usually associated to them (Blackburn et al., 2004).

Resource Use

While selecting the right tools for the job is important, it is also equally important to know how to use them correctly. A piece of software written in the lowest level language can still not be efficient if the developer does not know what they are doing. So, it is important to analyze the resources available when developing software and how their utilization can affect the end user experience.

There are a variety of resources at the disposal of developers that they can use including the CPU, encoders and decoders on it, RAM, the GPU, and other specialized parts, for example AI accelerators. While there are plenty of resources available, are they choosing them wisely? It is said that, just because one can does not mean one should, and it may be that developers are deciding to use recourses assuming that the software will be the main thing running on the computer when that is not compatible with the modern use case. Users now have multiple windows open with multiple tabs in each of a plethora of programs running at the same time either for increased productivity or of necessity (Saleem & Weiler, 2018). Do developers keep in mind or even know how to optimize software for energy efficiency? Some, like Pintor and Castor, say, “Developers currently do not fully understand how to write, maintain, and evolve energy-efficient software applications” (2017, p. 68).

It is thought that the quicker one gets something done the less energy one will be spending overall. Students are told something along these lines, where if they get their homework done the first thing when they get home, they will have the entire rest of the night, or if it is a Friday the entire weekend, to relax. The same logic has been applied to computers where the CPU will boost its frequency for a short time to get a task done

(Intel, 2024). One will often hear the fan ramp up for a second and then wind down when this occurs, especially in laptops. The problem with this is that while it is good in theory in practice it is very lax handing out that ability to programs causing it to be overused. The problem with this is that while it is good in theory, in practice the revving of the fan leads to increased power consumption.

Developers are optimizing for performance with the idea that because it gets done fast that it will consume less energy (Pang et al., 2016). But what is not considered is that these processes sometimes take a long time to complete, in addition to other processes running at the same time required by the modern workflow. This may cause the fan to ramp up for an extended period of time, thus wasting energy. Moving parts are one of the biggest culprits of power consumption, it is one of the reasons we have moved on from hard drives and CD/DVD drives, especially looking at its idle draw (Cho et al., 2015). Having that fan ramping up for one second is already consuming more energy unnecessarily; having it go on for an extended period of time greatly increases the battery consumption.

While it is true that the fan curve is set by the OEM of the laptop, it could not entirely be their fault. It could be a cautionary measure because of the lengthy processes that the OS thinks would take one second actually goes beyond that, greatly increasing the thermal temperature. The fan speed ramping up to account for this is meant to prevent thermal throttling.

Solutions to this would either lie with the developers or with the hardware OEMs giving their laptops a battery saving mode that while continuing to prevent screen brightness going over a certain limit and limiting background activity would also prevent the CPU from utilizing boost, thus staying at its base clock speed. A third option would

be to utilize the BIGlittle CPU core configuration, but with Intel's recent efforts, we have seen that is not a simple solution (Dornauer & Felderer, 2023). It may have to be a combination of BIGlittle and ARM found in today's modern smartphones, in Apples recent laptops, and Snapdragons upcoming ARM laptop chips for Windows to compete with them.

Chapter 3: Methodology

Based on the reviewed literature, resource use is something that can be hard to gather. Some require specialized hardware or software with access to the APIs the software uses to give accurate information and measurements. Thankfully, we have reached a point where popular software ships with developer tools that can help to identify what a web site or piece of software is targeting, the components that make it whole, and what exactly it is using on the computer.

Data was collected by using Google Chromes built in profiler, Mac OS's Activity Monitor, and Xcode Instruments CPU profiling tool. The built-in profiler helps to identify what CPU time is being spent on and what memory is being used for. The Activity Monitor is used as a basic comparison between the web apps and the native apps by looking at resource usage related to CPU, RAM, energy, disk, and network when only necessary programs are running to ensure untainted results. Xcode's Instruments CPU profiling tool gives trace of CPU usage over time that can be used to identify patterns in how the software is utilizing the hardware. Looking at the difference in statistics between the profiling Google Chrome itself when running a given web app and reading what the built-in profiler says the web app is using can give an idea of how much extra usage the browsers is tacking on.

First Collection

In Spring of 2023, research was conducted on how different websites use their computing resources, including what it spends its processing time doing as well as what it stores in memory. The computer used to conduct these experiments was a MacBook Pro Retina 13 inch from early 2015. The processor is a 2.9 GHz Dual-Core Intel Core i5. It has 16 GB of 1867 MHz DDR3 memory and Intel Iris graphics 6100 that is allocated

1536 MB of that memory. The computer is running MacOS Monterey, Versions 12.6.3–12.6.5, and the browser being used to run the tests is Google Chrome, Versions 110.0.5563.64–112.0.5615.137, using its own code profiler and other developer tools (Figure 1). Not updating unfortunately could not be avoided as small incremental updates could not be controlled and often made the OS (Operating System) lag or just broke Chrome. Although no difference was noticed between updates that the OS or Chrome could cause, data collection remained a consistent process throughout. During testing, all background processes are minimized, and any open processes will not use up resources that would affect the test results. To record the results, Microsoft Excel was used with multiple sheets for the various categories and OneDrive open to keep the files with the results synchronized and backed up. All settings on Chrome are default without any changes or added extensions to simulate how most people will use the browser. The computer was connected to the internet via Wi-Fi on the University of North Carolina Wilmington's (UNCW) network, though a few tests were done with the laptop connected to a Pixel 7 Pro acting as a Verizon 5G hotspot utilizing the campus of UNCW's newly installed 5G tower.

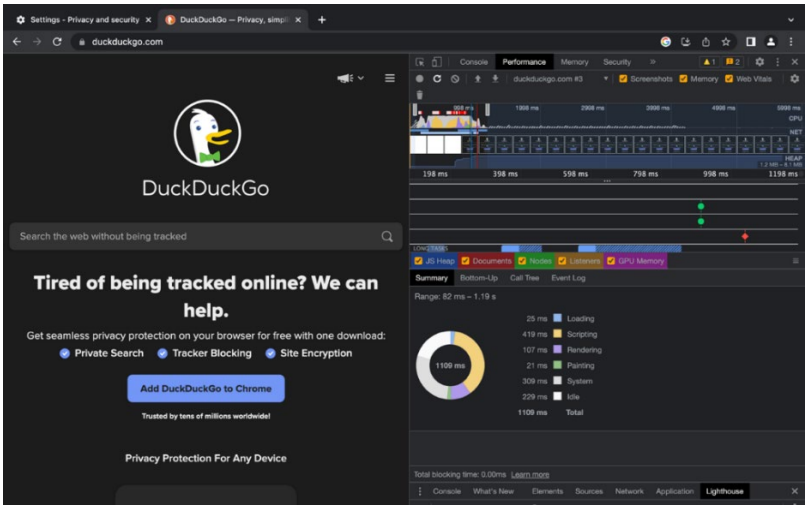

Figure 1: Google Chrome Performance Profile Example.

To access the developer tools used to get the profiling results either right click then click “inspect” from the dropdown menu or click the Google Chrome menu button, of three dots stacked vertically, hover on “More Tools” in the first dropdown menu and then click “Developer Tools” in the second dropdown menu. Both bring up sub windows that split the screen space between the current website and the developer tools. It starts on the console tab which will not be used for this research, the three main developer tool tabs used are “Performance”, “Memory”, and “Security”, that if missing from the top bar can be seen using the button with the two stacked arrows pointing to the right designated as “More tabs” that shows and allows one to click on those missing tabs.

To begin the first profiling test, head to the performance tab, once there, in a separate tab, open the Google Chrome setting page and select the “Privacy and security” section. Click on “Clear browsing data,” select the “Advanced” tab, click the “Time range” dropdown menu and select the “All time” option, this is to ensure that no precached data is manipulating the test results. Once cleared to run the first performance profile either use manual mode, the whole circle in top left corner of the performance tab, or auto-reload mode, the circular arrow right next to it. In manual mode once the button is clicked it will start recording processes happening for the current blank tab, “about:blank,” clear the address bar, then enter the website URL and then press Enter. It will load the website and then display it normally in the left sub window, once the website seems like it has loaded everything in, a good way to tell is, if in the bottom left corner there are no URLs popping up in a small black box, click the manual mode or rather record button again to stop the profiling. Auto-reload mode automatically starts profiling a website and stops when it decides that nothing else needs to be loaded in from the website given its time, between ten to sixty seconds depending on the length and the

content, to display the profiled data.

On top, a timeline showing what the processing time is used by color, and below it synchronized with the timeline are screenshots of what the tab looked like at a certain point of time during the profiling. Near the bottom is a sub-sub window with “Summary selected showing what the website spent processing time doing based on what range of the time is selected using two gray bars or what auto-reload mode selected for you. It is easy to get the range for manual mode to be remarkably close to what auto-reload mode would produce, if it is off by a bit, it will mostly add idle time which is not too important to this research.

The Summary tab has hollow color-coded pie chart with the total time range in the middle, including idle time, and to the right is the time in milliseconds are type of process took, the color of the process, and then the name of the process all stacked on top of each other. The types of processes and their colors are “Loading” as blue, “Scripting” as yellow, “Rendering” as purple, “Painting” as green, “System” as gray, and “Idle” as white. “Loading” is the time spent loading and parsing HTML, CSS, and JavaScript. Scripting is the processes enabled by JavaScript.

Rendering and painting are displaying HTML and CSS from code to visuals. “System” is calling to use system resources, like the camera or microphone, and built-in system libraries like graphics APIs (application program interfaces). These results are then copied to the appropriate cells in Excel. The performance profile is saved by pressing a button with an arrow pointing down at a thin horizontal line. Those results can be viewed again later by pressing the button next to the download button, the one with the arrow pointed up, and then selecting the file on the computer.

The second part of the profiling is the memory using the “Memory” tab. It will

show three different options for profiling, for the most part “Heap snapshot” with the checkbox “Include numerical values in capture” under it checked, will be all that is needed. To begin the snapshot, press the blue button at the bottom of the developer tools window “Take snapshot” which will then begin the snapshot taking as low as a few seconds up to a minute based on the content. It will then show the heap snapshot selected in the left side bar with the result shown to the right.

The results needed are not in the “Summary” tab in the sub-sub window so we will change it to “Statistics” in the dropdown menu, which gives a familiar looking hollow color-coded but not complete pie chart. To the right of that pie chart is the amount of memory, in kB, taken up by a certain aspect, the aspects color, and then the aspects name all stacked on top of each other. They are “Code” in red, “Strings” in green, “JS arrays” in blue, “Typed arrays” in yellow, “System object” in purple, and the combined total amount of memory at the bottom. Code refers to the instruction being sent to the CPU. Strings refers to text and other objects to be displayed. JS arrays are the array structure JavaScript uses. Typed arrays are raw memory, or rather binary data, so they are super-fast and do not have to be converted to another representation like regular arrays. System objects memory is used to maintain processes that call on built-in system libraries, like those that process audio and video.

For websites that have a continuous amount of content being profiled, like video hosting, two heap snapshots will be recorded one before and one after the video content has loaded in. During the playback of the video an “Allocation instrumentation on timeline” profile will be recorded with the “Record stack traces of allocations (extra performance overhead)” box checked. This will be a manual recording of memory usage with a timeline showing when new memory is used and what that is. Just like the

performance profile timeline one can select a time range to see only what was loaded into memory at that time. This is useful for video playback because it can show how the website decides to load in video, medium sized chunks at regular intervals, small chunks all throughout the run time, or one or a few large chunks. On the left side bar, downloading that heap snapshot or allocation profile can be chosen. Like the performance profile a load button can be used, this time at the bottom where the button to begin the profiling was, to view that data again later. These results were taken and then put in the appropriate cells in Excel.

Lastly is the Security tab, which does not need to be started manually, but the develop tools sub window must be opened before the website is loaded to get the results, it does not matter what tab. Once in the security tab, it starts with the "Overview" on the left side bar selected, showing to the right the "Security Overview." This shows if the HTPPS is valid, if the certificate is valid and trusted, details on the certificate, if the connection is secure, details about how the connection is secure, if resources are served securely and if there is any extra info on that. This info would be put into Excel. The other info needed does not require another sub menu, but is listed on the left side bar under the overview as "Main origin," "Secure Origins," and if there are any "Unknown / canceled" origins. These are written manually into a separate Word document, one document for each website category, except for when the number of secure origins started exceeding around fifty plus items and could not be effectively done in the time given.

Once that started to happen, screenshots were used to capture the origins. These files, once created, were put into the same folder as the websites profiling data files. The data put into Excel would be the main origin URL, the number of secure origins and the number of unknown or canceled origins.

The final pieces of data in Excel are done before the rest, but since it is not the focus, it is pushed to the right behind all the other data. Using the Ookla Speed test app from the Mac App Store, the date was recorded (YEAR_MM_DD), network, download speed (Mbps), upload speed (Mbps), Ping(ms), Jitter(ms), and Packet Loss (%). Most profiling was done using the UNCW (University of North Carolina Wilmington) Wi-Fi network with an average speed of ~120 Mbps, ping of ~17 ms, jitter being zero, and zero percent packet loss.

For a few profiles, a Pixel 7 Pro's hotspot capabilities using the Verizon 5G network were used because certain websites were blocked. The speed using the phone as a hotspot ranged from about 50-100 Mbps down, 3-10 Mbps up, 37-53 ms ping, 11-14 ms jitter, and 0% packet loss. The download speed is the most important which is adequate, and while the upload speed is now much lower it does not really matter because all we are uploading to the internet are requests to load websites which are much smaller than what we a receiving. That is not as much as what the websites send to the computer to be stored in RAM.

First Websites

The websites chosen to be profiled can be put into multiple categories that will theoretically have different usage and levels of usage of the computer hardware. The categories are search engines, news, online shopping, social media, and video hosting. These categories were chosen because they all cover different use cases and potential resource needs.

Search engines take the input and compute what it thinks the user is looking for, and then returns that to the user in some form whether it be single text hyperlinks or multimedia of some kind. Search engines are also a part of every other category to find

content particular to that site. Lastly, search engines are what people use to get to all other websites, including other search engines.

News and online shopping are both categories of sites that are trying to sell the user something explicitly. They are going to potentially use system resources to display items it wants to promote, track users' preferences for items, and serve better suited items to them.

Social media and video hosting both delve into user generated content, images as well as text posts for the former and for the latter the addition of video playback. Social media will potentially not be as resource heavy as video hosting sites because of the latter's use of video decoder and if those are not supported by the hardware or software the CPU will be used consuming much more resources. Though social media does include videos but because they are usually short in length it does not matter as much.

The websites will all be profiled at least once on their home page and a second time if they have any comparable internal pages that visually or theoretically would have a big enough content change to warrant a profile. The landing pages profiled will be as similar as possible, for example, all search engines will search "google" when profiling their landing page.

For search engines we have Google, Bing, Yahoo, Yandex, DuckDuckGo, DuckDuckGo HTML, DuckDuckGo LITE, SearX, and Swisscows. Google is the primary search engine that people use and will use many resources due to the company being a data collection powerhouse. Bing, owned by Microsoft, and Yahoo, owned by Verizon, are the two main competitors to Google that are backed by big Corporations. Yandex is a search engine originating from Russia taking on the likes of Google, it is likely this one will be giving information to the Kremlin. DuckDuckGo is a pro-privacy and generally

“pro-free speech” search engine, which has three different versions to choose from, two of which being non-JavaScript versions. SearX is a search engine that is an aggregator, which means it pulls from a multitude of search engines to combine into one and is hosted across many domains. It is possible to set up one’s own instance of SearX. Swisscows is a search engine that is focused on education and blocks any “inappropriate” search results, a dedicated Google Scholar.

For news we have NBC, NYTimes, Forbes, CBS, CNN, Fox News, and RealClearPolitics. There is no main website that people go to for news here, besides Fox News for the right side of the aisle, and RealClearPolitics is just an aggregated site of news from the previously mentioned sites and more. There is also Apple News and Google News, but those aggregators are not really meant for desktops and function better on phones, those will not be tested.

Online shopping includes Amazon, Walmart, eBay, Etsy, Alibaba, and Wish. Amazon is the main online shopping site that people use. Walmart was selected because a short-lived competitor named Jet was integrated into Walmart’s site. eBay is the go-to site for buying and selling used goods. Etsy is focused on the sale of handmade goods. Alibaba and Wish can both be described as value product sites where stuff can be bought for cheap but often has fraud listings.

Social media being made up of Twitter, Gab, Rumble, Getter, Reddit, and 4chan. These sites were chosen because of the ability to view them without a profile, one of the reasons Facebook was not included. Twitter, while the biggest, is not as far ahead of its competitors compared to the other categories. Gab, Rumble, and Getter are a group of Alt-Tech sites created due to restrictions made by Twitter and Facebook alike. What Alt-Tech is will be explained more later.

The last main group, and what was the most demanding, is video hosting which has YouTube, Dailymotion, BitChute, Odysee (LBRY), and Rumble. YouTube is the biggest video hosting site and to some is also considered the second largest search engine, it is owned by Google. Dailymotion was a competitor to during the late 2000s and early 2010s but has not gotten more popular since and has been overshadowed by YouTube. BitChute, Odysee, and Rumble can be described as another group of Alt-Tech sites, created in part due to increased restrictions on the content that is allowed on mainstream sites like YouTube.

So, what exactly are these alternative technology sites previously mentioned? These sites were created to be alternatives to user-driven content sites, categories covered being social media, video hosting, and in a sense search engines. The market for these sites was not created overnight but over the years. One thing that created them was the abuse of copyright laws and DMCA. People often used copyright material in their videos whether that be music, clips from tv shows or movies, pictures, etc. And while people did often upload the original content unaltered these chases could have been handled on their own but instead the websites decided to do blanket demonetizations or even removal on that content which hurt the user base. This continues to be an issue today where content that should be protected under fair use is taken down unjustly.

The other factor that became a catalyst for their birth was political censorship. It started out with just the “fringes” of the political spectrum, but overtime quickly started covering viewpoints and statements in the mainstream. Because of this the inclusion of Alt-Tech sites may be looked down upon by some but they provide an excellent contrast. Some of these Alt-Tech sites use frameworks that are different from the mainstream tech sites, some part due to natural innovation and some forced innovation due to them not

being able to use those established frameworks. One of those alternate frameworks would be LBRY, what Odysee uses to power its website, which is based on Blockchain, uses Web 3.0, and is a network that other sites can build from to have the exact same content.

Second Collection

For the current research, the hardware will be the same and will keep the same measurement tools in addition to new ones. Though the OS and software used are now of a newer version. The computer is running MacOS Monterey, Version 12.7.4, and the browser being used to run the tests is Google Chrome, Versions 123.0.6312.123–123.0.6312.124. Also, when profiling the websites the manual timer will be used for all of them now instead of using the automatic setting for most of them like last time. This is because the automatic profiler can sometimes mess up its time window it selects when it performs the profiling. Doing the manual mode will ensure that does not happen because manual adjusting the profile time window to want processes are visible will only have deviation for the idle time which is not the focus and has proved to not be important. Note that this fact does not make the previously collected data useless as the time window was adjusted in case it did it wrong, moving it over to fully manual makes the process uniform and in a way easier. The allocation timeline will also not be done for the second collection due to it not having much use in the first; Instead, heap snapshots from when the video has completed playing will be compared.

Between the first data collection and the new data collection the existence of Mozilla Firefox's own performance profiler tools has become known. While that could potentially be useful in resource usage between browsers on particular websites, it would not be useful for the aims of this particular research. That is because since the old and new data will be compared to each other there are potential differences with the profiler

results that cannot be directly compared, combined with the fact that accurately profiling these websites how they were a year ago is not possible. So, to have an accurate comparison Google Chrome must be used again with the same websites.

The Mac OS activity monitor allows the viewing of data related to CPU, Memory, Energy Disk, and Network usage (Figure 2). Xcode process profiling tool with Instruments and Samples can help to expand that. The reason that these two are used compared to other possible options is that since they are developed directly by Apple, they should be the most accurate with their results and their resource usage when profiling, so it doesn't mess with the website or application of interest. All applications on the computer that are not necessary and open only the ones that are needed, potentially restarting the machine with nothing opening at start up. Note would be taken of what processes are already open during the initial start-up and the processes added when the applications necessary are opened to complete the data collection, so it is known what process needs to be profiled.

The process of collecting the data from the activity monitor goes as follows. It starts on the CPU page, working through each of the other tabs once the data is collected from the current. When collecting data from the native apps it is easy to identify what process is the one that should be targeted because there are only a couple subprocesses or none. Data collection though activity monitor for Google Chrome was not as simple. Chrome uses multiple helper sub processes that run within the main process. The problem this causes when it comes to data collection is that data points related to a specific process handling a window or a tab of the browser do not count towards the main process in the activity monitor. So, selecting the helper process that is handling the window or tab running the website is needed. Once it is selected a screenshot is taken of the activity

monitor stats at that point in time a recorded to an Excel sheet. This is done again and again for each tab of the activity monitor.

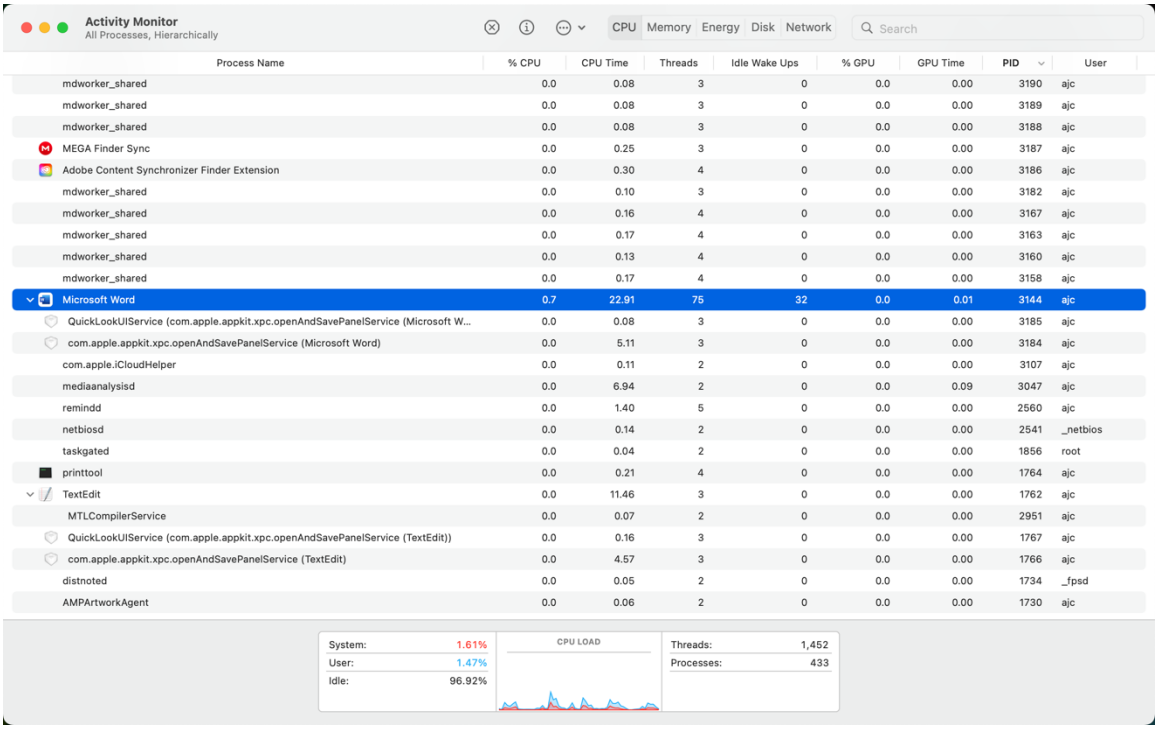

Figure 2: Activity Monitor Example.

The data collected from the activity monitor are as follows, For the CPU the number of threads is taken (Hoakley, 2022). The threads are the number of instructions waiting to be executed for a process. The CPU percentage utilization was not used because of its drastic variability from one second to the next.

For Memory, the memory, the physical footprint (PF), peak physical footprint (PPF), real memory, real private memory, real shared memory, purgeable memory and (VM) compressed memory. The PF and PPF are taken from running a sample process, PF should be around the same as memory and the PPF if the max amount of memory it ever used. Real private memory is “physical memory being used exclusively by that process for its own code and data” (Hoakley, 2022). Real-shared memory is “physical memory being used by that process which is shared with other processes” (Hoakley, 2022).

Purgeable memory is “physical memory being used to store items that could be cached to disk if necessary to free up physical memory” (Hoakley, 2022). Compressed memory is “physical memory which hasn’t been used recently, so its contents have been compressed to save space” (Hoakley, 2022).

For Energy, the energy impact, 12-hour power if applicable, and if App Nap is enabled and if it is preventing sleep. For Disk there is the number of bytes written and read. Lastly for Network there are the sent and received bytes as well as packets.

The Xcode instruments CPU Profiler provides the one thing that the activity monitor does not, a timeline. With this a trace is taken to allow the analysis of CPU usage across time. This can identify how often the CPU is being tasked to complete processes for an application and how hard it stresses the CPU.

To record this timeline the Xcode app first has to be launched, then right the icon in the app bar to bring up the pop-up menu where “Open Developer Tool > Instruments” (Figure 3).

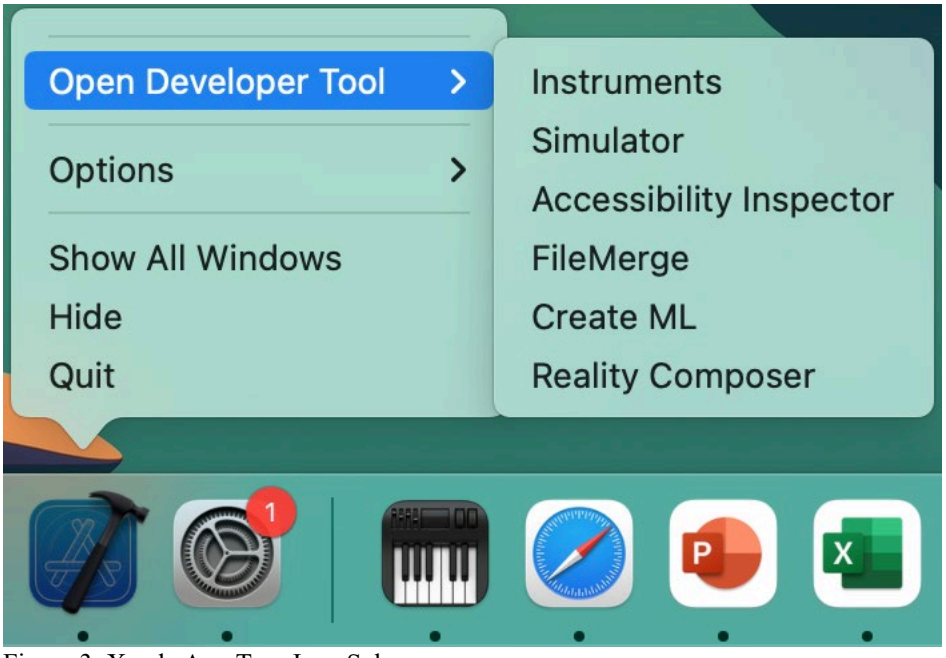

Figure 3: Xcode App Tray Icon Submenu.

That will open a pop-up window with a selection of diagnostic tools, of which the "CPU Profiler" will be selected (Figure 4).

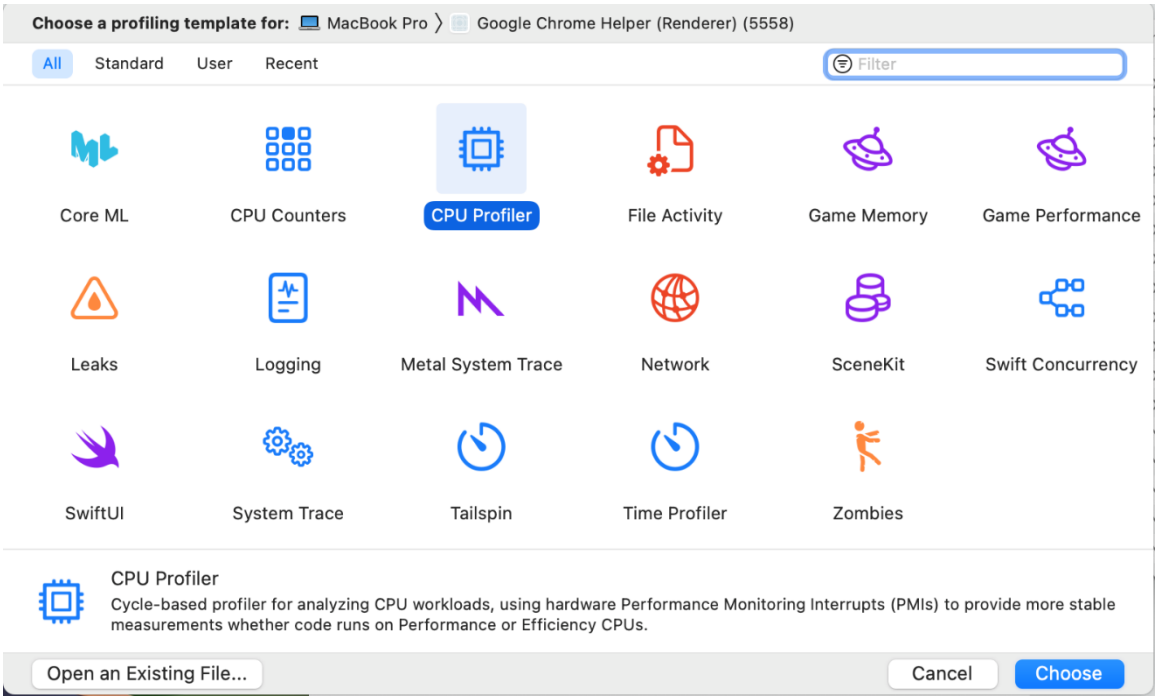

Figure 4: Xcode Diagnostic Tools Pop Up Menu.

A window with three timelines for CPU Profiler, Points of Interest, Thermal State, and once profiling starts the process itself (Figure 5).

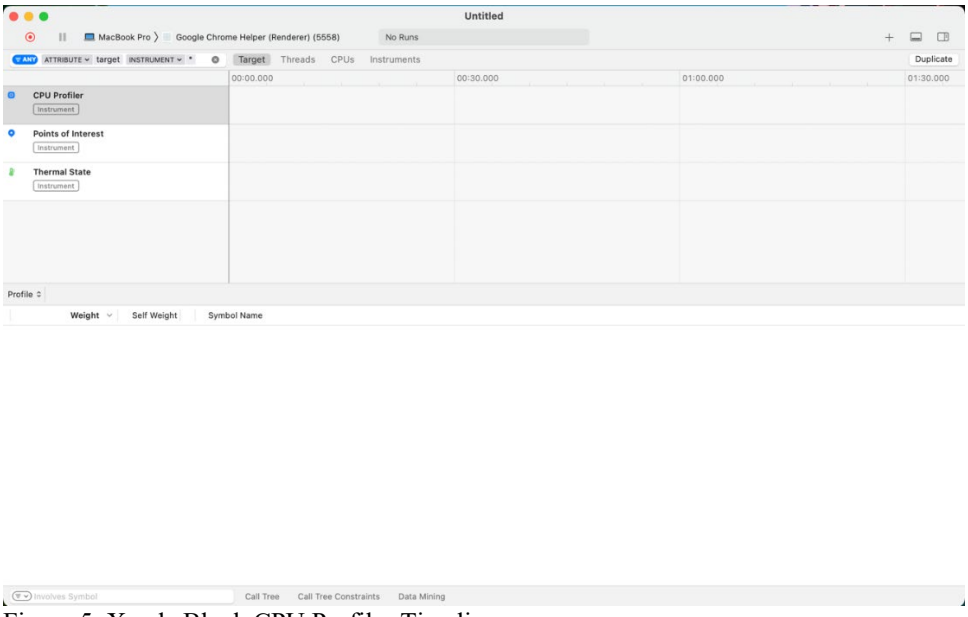

Figure 5: Xcode Blank CPU Profiler Timeline.

There is a top bar with the currently selected process left of the middle. Clicking this will open a drop-down menu with some quick options of what process to profile, recent processes will usually show up here (Figure 6).

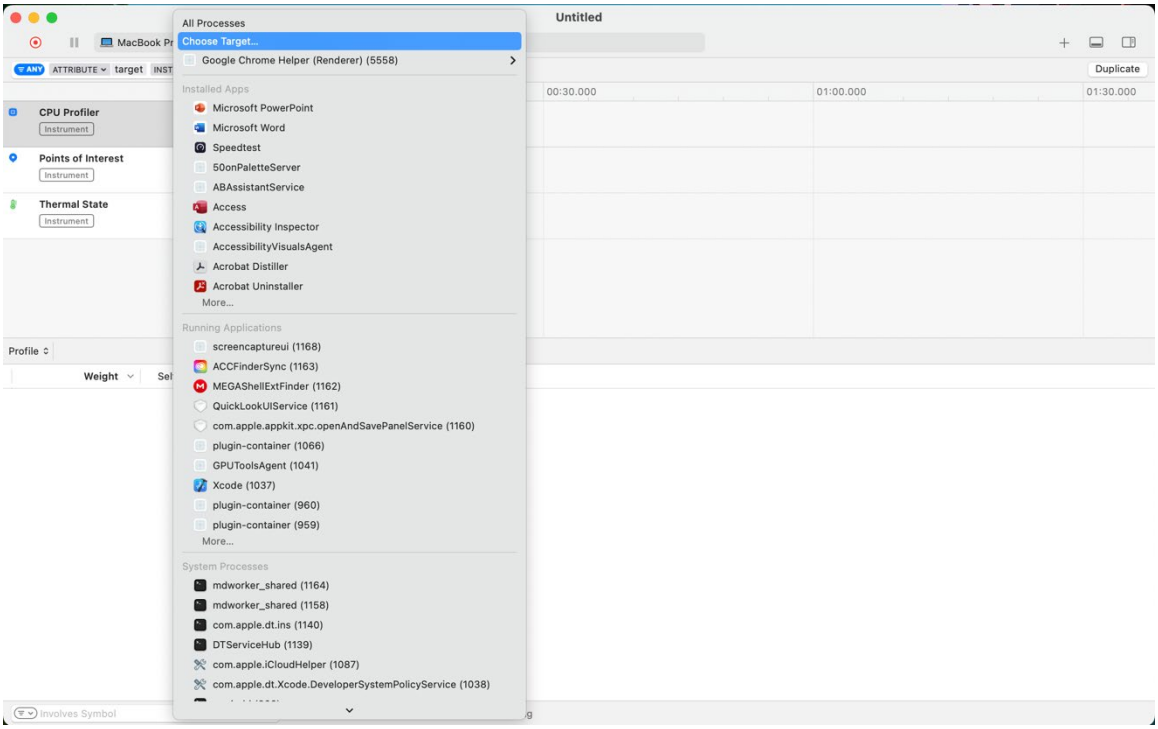

Figure 6: Xcode CPU Profiler Timeline Process Selector.

If for some reason they do not, “Choose” will allow the selection of a specific process. With the process selected then the top left red record button can be pressed to start the profiling and then again to stop. Lastly a screenshot is taken of the timeline and then using “File > Save As” is archived and can be loaded up for future viewing (Figure 7).

Additional Websites & Apps

The current set of websites includes the previous set with the data collection redone. The new sites added include the web versions of the MS Office suite. They are then compared to their native counterpart on Mac OS by using the activity monitor and the Xcode process profiling tool. Using this on Google Chrome as well will potentially

show the added resource use caused by the browser vs the web app itself. It could also be used to compare the combined resource use of the web app and browser to the native version. Theoretically we will see that the native version has less resource use or at least performs better when having large files open.

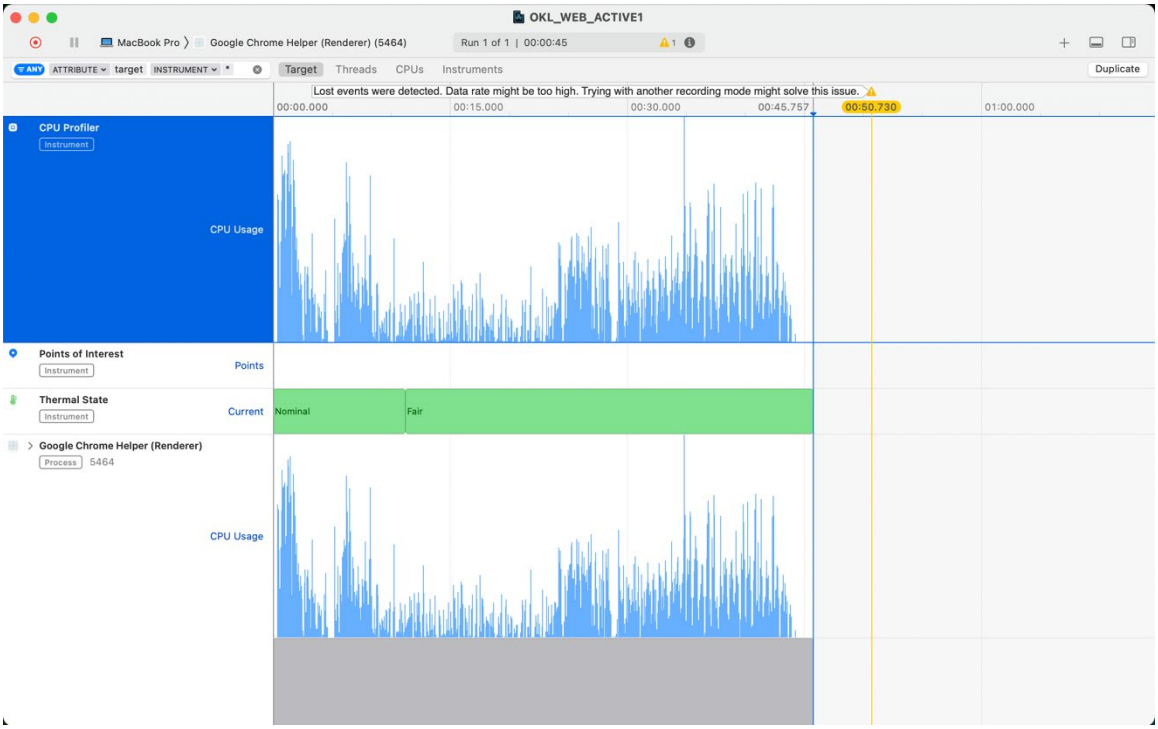

Figure 7: Xcode CPU Profiler Timeline Example.

One category of websites that will not be covered in this research but can be explored more in the future is institutional and organizational websites. These types of websites would have no consistently changing user or website generated content instead being static and not meant for heavy traffic, just for information. Though theoretically because of this they would not be resource heavy and are not part of a user's average workflow which would not reveal anything noteworthy.

Chapter 4: Timeline for Completion

To get an idea of the time it would take to do the research, the profiling and data collection for the search engine category was redone ahead of time. Because of the familiarity with the process and the fact that a template now existed that could be used again it only took two hours. Table 1 shows the categories of sites, the specific sites, the amount of time estimated to complete the data collection, and how long it actually took.

Table 1.

Categories, Sites, and Estimated Time for Completion

Category	List of Sites/Apps	Estimated Time	Actual Time
Search engine (Landing & Internal Page)	Google, Bing, Yahoo, Yandex, DuckDuckGo Java Script (JS), DuckDuckGo HTML, DuckDuckGo Lite, SearX, and Swisscows	2 hours	2 hours
News	NYTimes, CNN, Forbes, Fox, and RealClearPolitics	1 hour	1.5 hours
Online shopping	Amazon, eBay, Etsy, Walmart, Alibaba, and Wish	1.5 hours	1.5 hours
Social media	Twitter, Gab, Getter, Reddit (New), Reddit (Old), and 4chan	1.5 hours	2 hours
Video hosting (Landing & Internal Page)	YouTube, Dailymotion, BitChute, Odysee, and Rumble	2 Hours	3 hours
Web apps vs. native	MS Office Suite(Word, Power Point, Excel), Ookla Speed Test	~2 hours	3 hours

Overall, the time taken to do the profiling was greater than expected. The search engine category and online shopping categories were the two that took the estimated time. For the news category there were certain cites that took a sustainably longer time to load compared to the rest. Video hosting took longer because of ads that played before the main video and could not be avoided. The new data collection using the activity

monitor and Xcode took longer than expected. Some of this has to do with how activity monitors data and is always changing. A screen shot was needed to collect data at a certain point of time for an application. There were cases collection was not fast enough that the whole process for the website or native application had to be redone. Xcode was completely new compared to activity monitor where there was at least some familiarity. The collection in total for that category took about three times longer than expected or four hours extra.

The results for the profiling and data collection will be looked at in terms of what is taking up CPU time and Memory resources. The finds will be displayed for each category, new and old, with bar charts for the CPU time and memory usage (Figure 8). To compare, there will be charts showing the difference between the old and new data that could be used to show website improvement, regression, or a mix of the two over the last year.

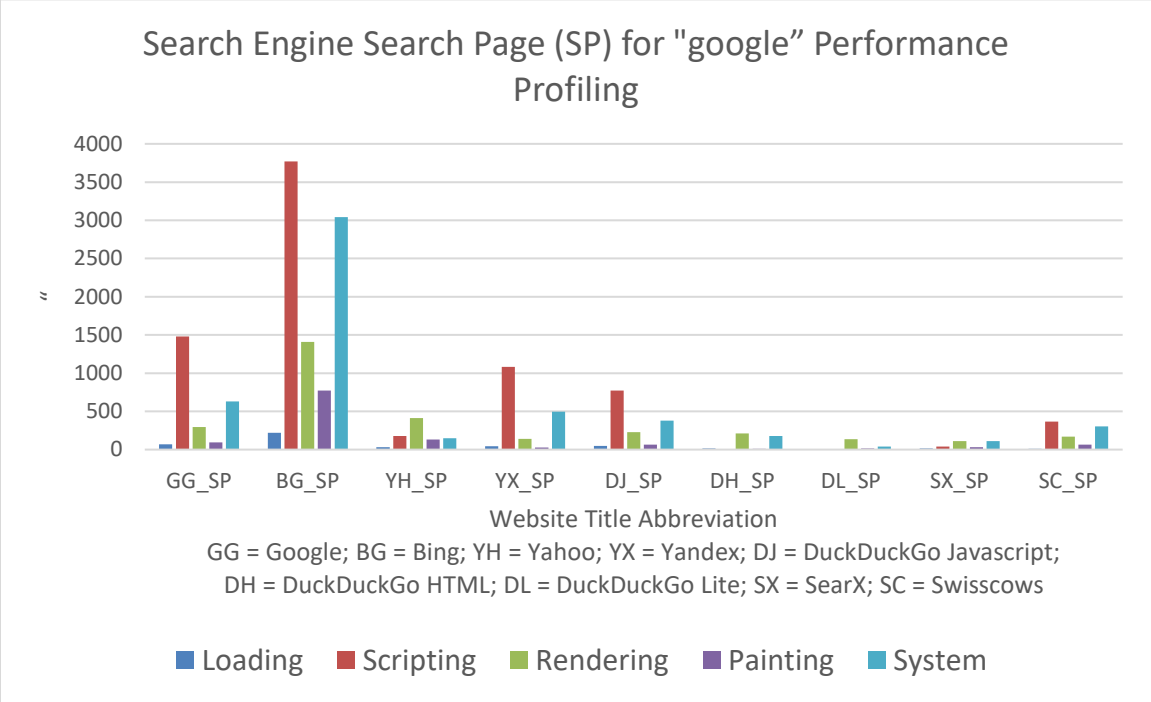

Figure 8: Example of Collected Profile Data.

From these recommendations will be made to not only developers, pointing out the resources they need to optimize the use of as well as point to good examples of the websites profiled if any, but users to recommend any alternatives to sites and apps that would minimally impact their work as well as provide a performance benefit.

Chapter 5: Results and Discussion

Search Engines

First Collection. The results of the previous research in the search engine home page performance profile give us an immediately visible contrast into these kinds of websites (Figure 9). The first thing to note is that Google does not seem to chart at all here, but since this is Google's own browser it is possible that it is already preloaded into Chrome so there is no time spent on anything. Second, when it comes to profiles six through eight, eight which is SearX did score on the measurements, but it is so low compared to larger ones that they do not visibly chart, and six as well as seven, being DuckDuckGo HTML and LITE, while like the later do not chart any scripting, that because there is nothing to script.

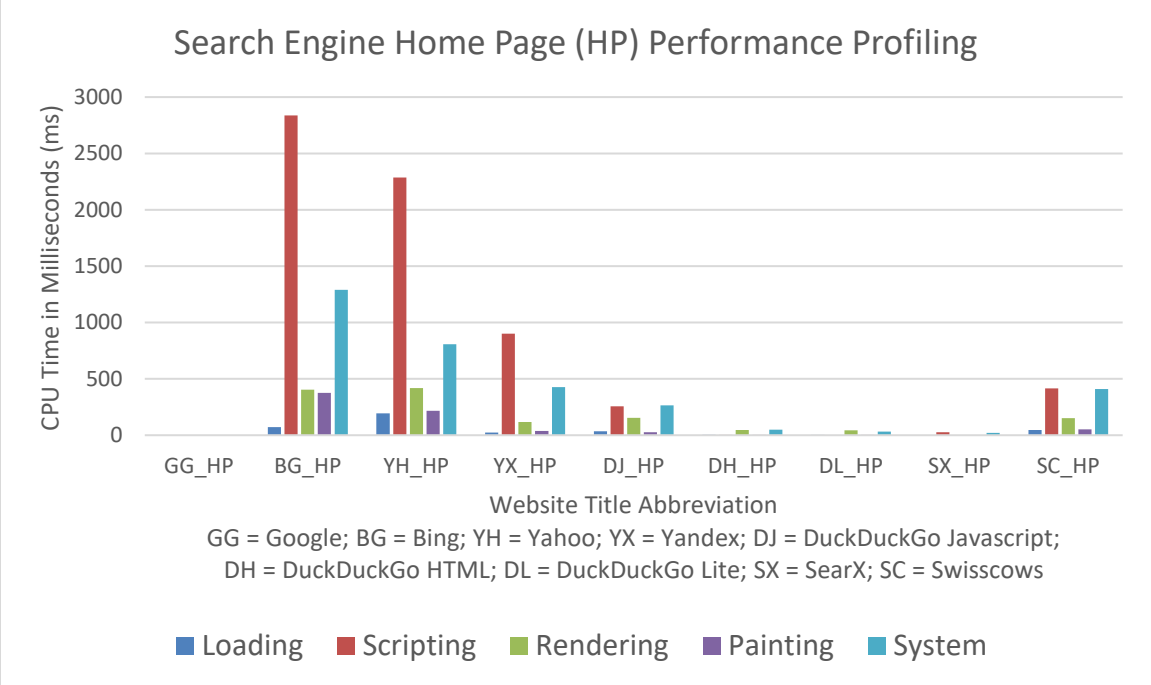

Figure 9: First Collection Search Engine Home Page Performance Profiling.

With those important pieces of info out of the way, the results are predictable. All websites took extraordinarily little time to load, with Yahoo taking the longest at just 195

ms. Both Bing and Yahoo take the longest amount of time overall, with Bing being slightly higher with scripting and system CPU time use. Yandex used around one fourth of the CPU time that Bing and Yahoo used, making it a smaller version of the three mainstream search engines. With Yandex being relatively new and not having any dedication towards privacy, it is possible one day it could catch up to the big guys in its CPU time usage. The rest of the search engines, including all DuckDuckGo versions, SearX, and Swisscows are not CPU time light. While you can say that Swisscows is the one spending the most overall CPU time followed by DuckDuckGo JS, at this point where we are getting to a total CPU time of around one second. Using search engines with a sub one second CPU time is ideal in some use cases but does come at the cost of looks and functionality, which of search engines are varied.

Google is the simplest, it is a blank background with the Google logo and the search bar which when something is initially typed autocompletes the query to the best of its ability. Most search engines, except those with no JavaScript, can do this. Bing adds onto this by making their home page a bit flashier and introducing a news section. Yahoo takes this and runs with it; the entire page is filled with adds for news stories. Because of this Yahoo's number of secure origins are around triple to quadruple the amount Google and Bing had while dwarfing the rest of the search engines which on average had around one to two secure origins, maybe a few more in Yandex' case. Speaking of which had a remarkably similar home page to Bing if talking about components. DuckDuckGo JS is most like Google along with SearX. DuckDuckGo HTML and LITE are even more bare, with a logo, a search bar, and a begin search button, all without auto search query completion as well. Swisscows home page is quite different and is in a way most like Yahoo's, in an effective way though. The search engine has the search bar at the top and

then below it is all other information about the search engine and the company including new, updates, and advertisement not for other companies' products but their own.

The results of the search engine home page heap snapshot profile are a similar story (Figure 10). Yahoo takes up the most, because of all the ads it loaded up. Bing being the second largest is interesting, because while it is smaller it is not as small as one thinks it would be considering it has a fraction of the ads. Google is even more surprising, while again it is a good chunk lower it still is using lots of memory for code, given Google's advanced algorithm this may not be much of a surprise. Besides Swisscows with a notable bump in the memory it uses for coding, the rest of the search engines use such a small amount of memory it is not worth considering.

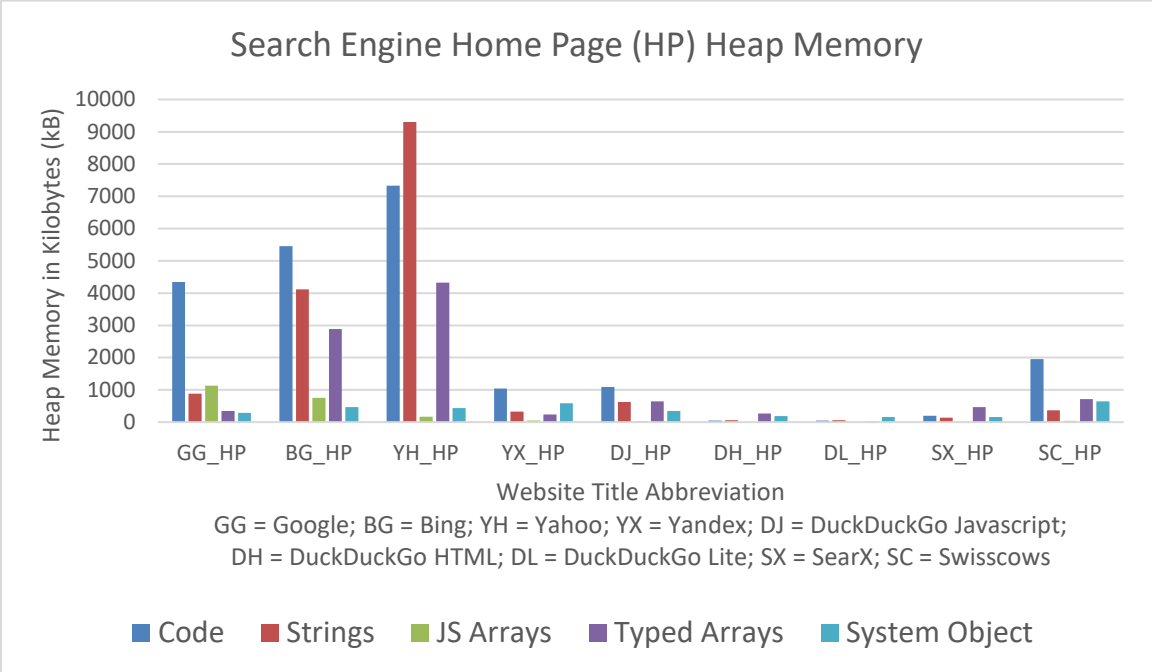

Figure 10: First Collection Search Engine Home Page Heap Memory.

The results of the search engine search page performance profile show comparable results to the home page one but a bit more interesting when we look at the details (Figure 11). Google does a lot of scripting, again most likely because of their

advanced algorithm from being in the game for so long. Bing uses CPU time in scripting, rendering, and system quite equally. Yahoo, Yandex, and DuckDuckGo JS all have a similar pattern of CPU time usage. The rest of the search engines are even lower, making performance worries while searching in multiple tabs an afterthought. Swisscows might be using up a decent number of resources for a privacy focused browser because of its filter which blocks “inappropriate” content.

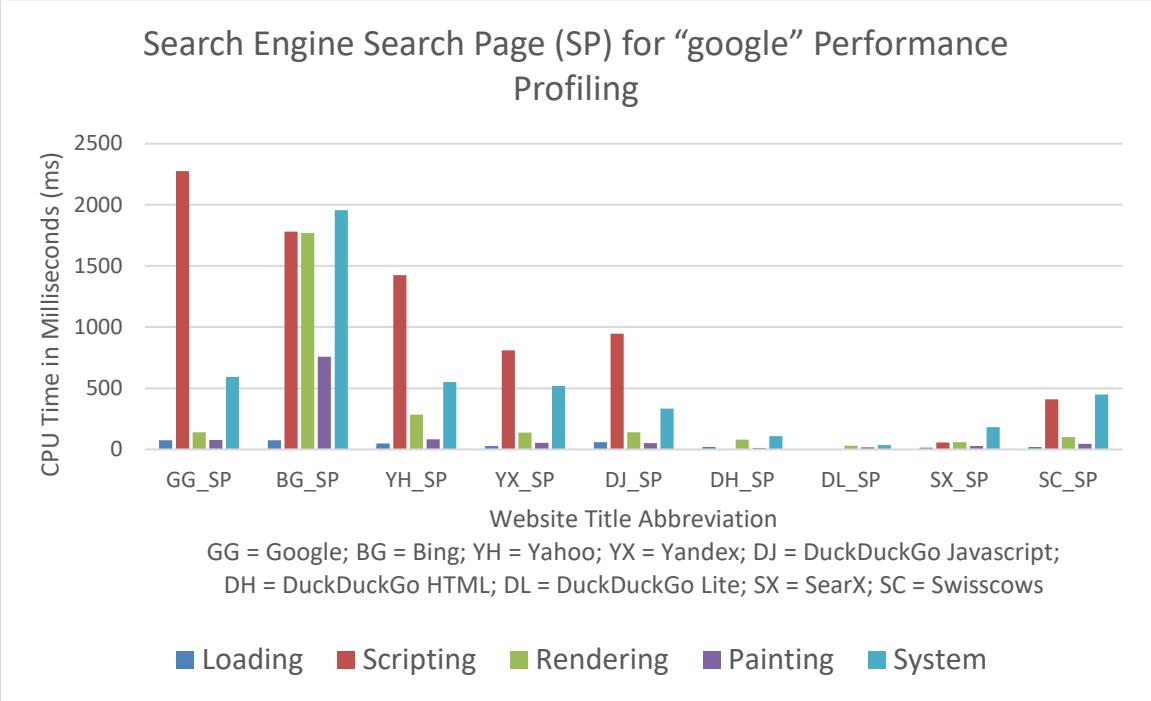

Figure 11: First Collection Search Engine Search for “google” Performance Profiling.

The results of the search engine search page heap snapshot profile show a shocking result where Google takes it away in terms of memory usage and the other mainline sites, while using more than the bare bones search engines, are dwarfed by google (Figure 12). Besides Yahoo using a suspicious amount of memory for system objects the rest of these are incredibly low and should not need to be worried about.

Second Collection. The new data collection reveals that unlike before Google is

actually charting for its home page. This could mean that Google not giving data before for the CPU time was an issue with that version of the developer tools or they stopped preloading it into the browser. Bing sees an increase in all aspects of CPU time but especially loading which it spent practically nothing previously doing which could be to the new Copilot implementation (Figure 13). Yahoo overtakes Bing with the most time spent scripting, with the latter now taking an equal amount of time to script and load possibly due to its copilot implementation. Yandex seems to be lower across the board possibly due to the US side of the business splitting ties with Russian because of the Ukraine war. Lastly memory usage seems to have increased for Google, Bing, and DuckDuckGo in code (Figure 14). Interestingly it seems that the memory usage for strings in Yahoo has decreased since the last time being lower than its code usage which also lowered.

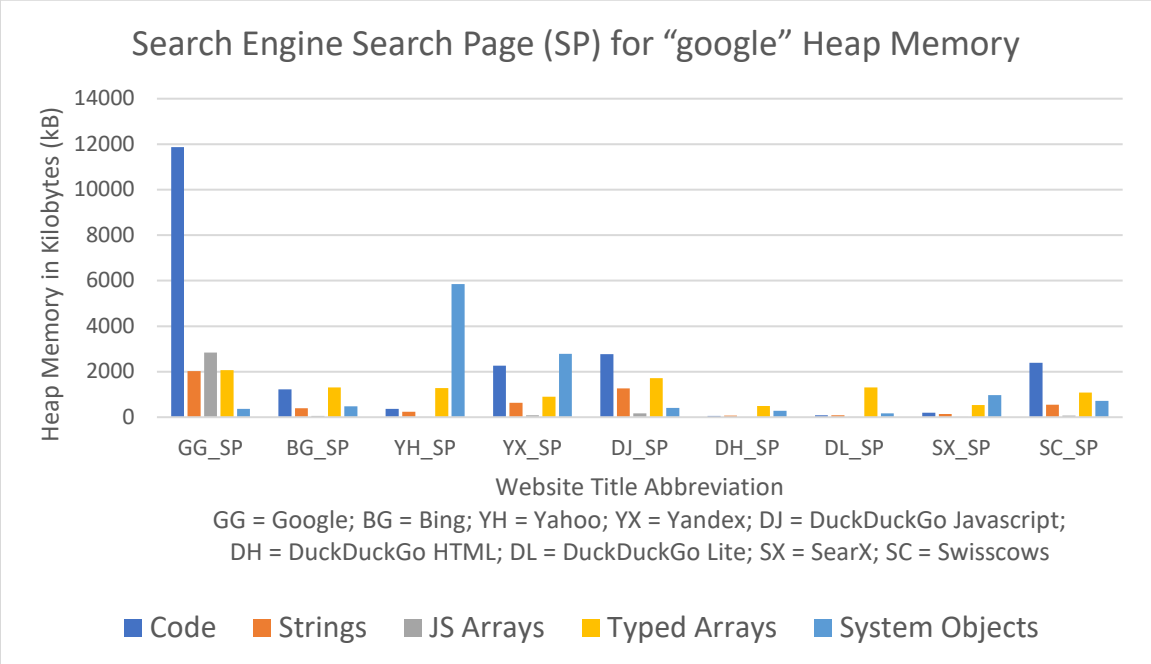

Figure 12: First Collection Search Engine Search for “google” Heap Memory.

The only noteworthy results for the search page profiles are for the big three.

Yahoo is way down on its scripting and system usage being lower in time than DuckDuckGo JS (Figure 15). Google noticeably cut a chunk out of its scripting time but everything else seems the same. Bing unfortunately seems to have greatly increased in scripting and system usage. On the memory side Google sees a decrease in code but an increase everywhere else (Figure 16). Bing has greatly increased in memory usage all around as has DuckDuckGo JS and then Swisscows. Yahoo has seen a bit of a tradeoff drastically cutting its system objects but increasing its typed arrays. Yandex also seems to have cut its system objects usage increasing its code usage.

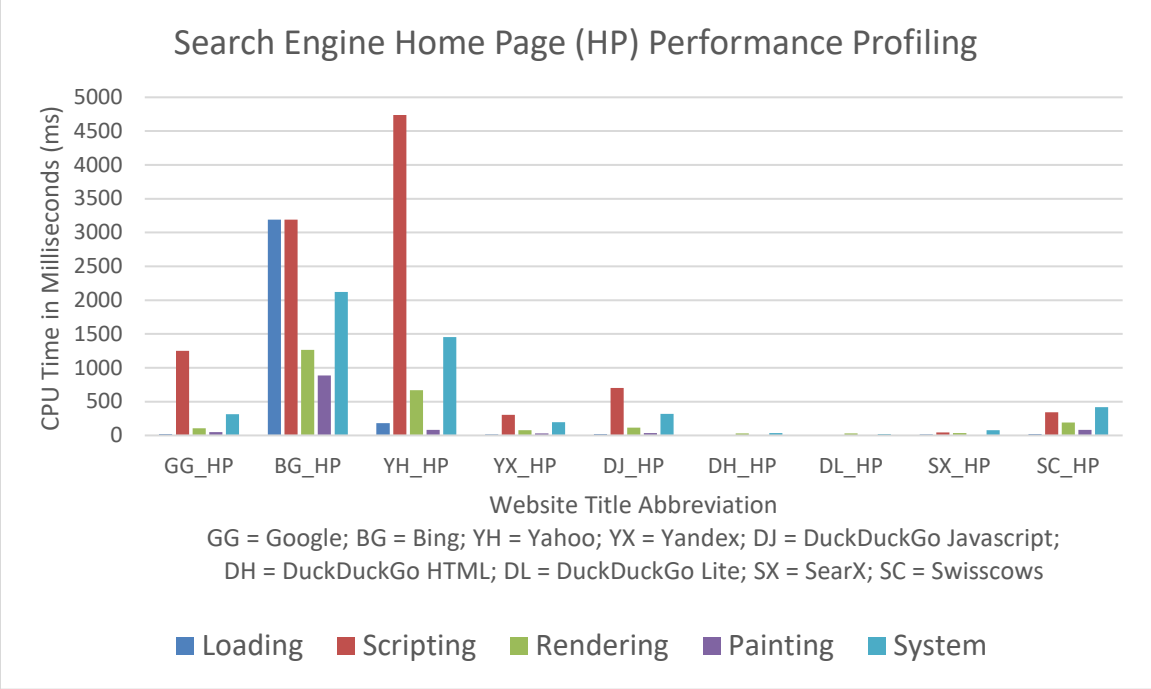

Figure 13: Second Collection Search Engine Home Page Performance Profiling.

First vs. Second Collection. Figures 17, 18, 19, and 20 show the differences between the first and second collections for the search engine category indicating improvement or regression.

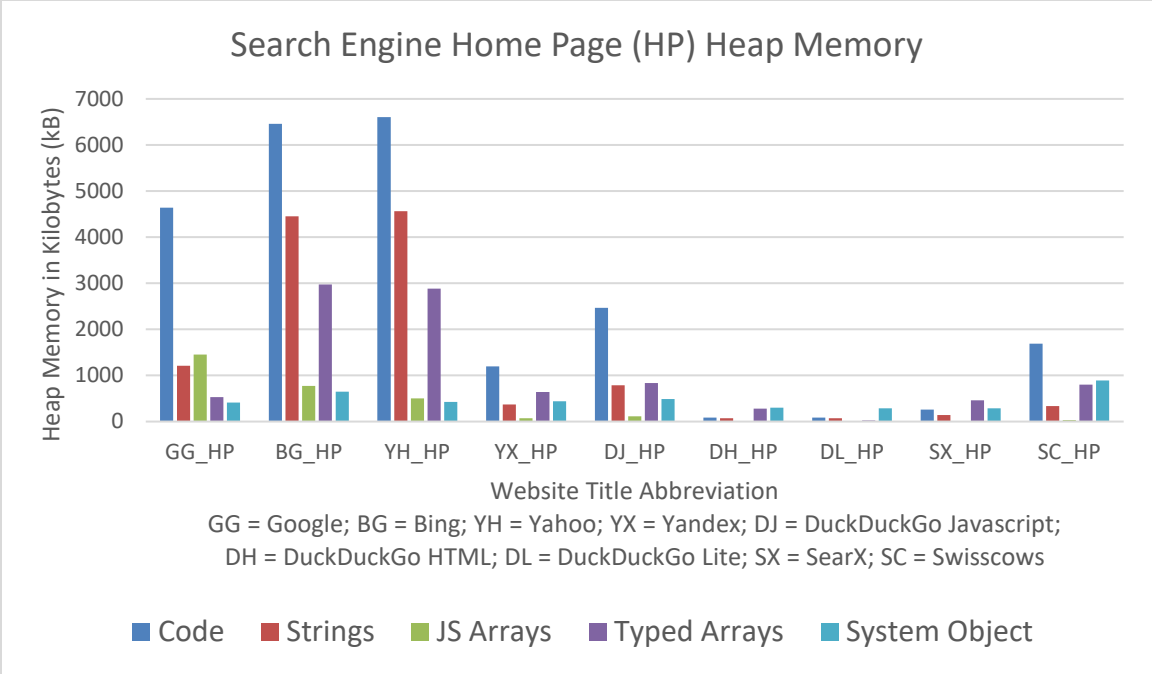

Figure 14: Second Collection Search Engine Home Page Heap Memory.

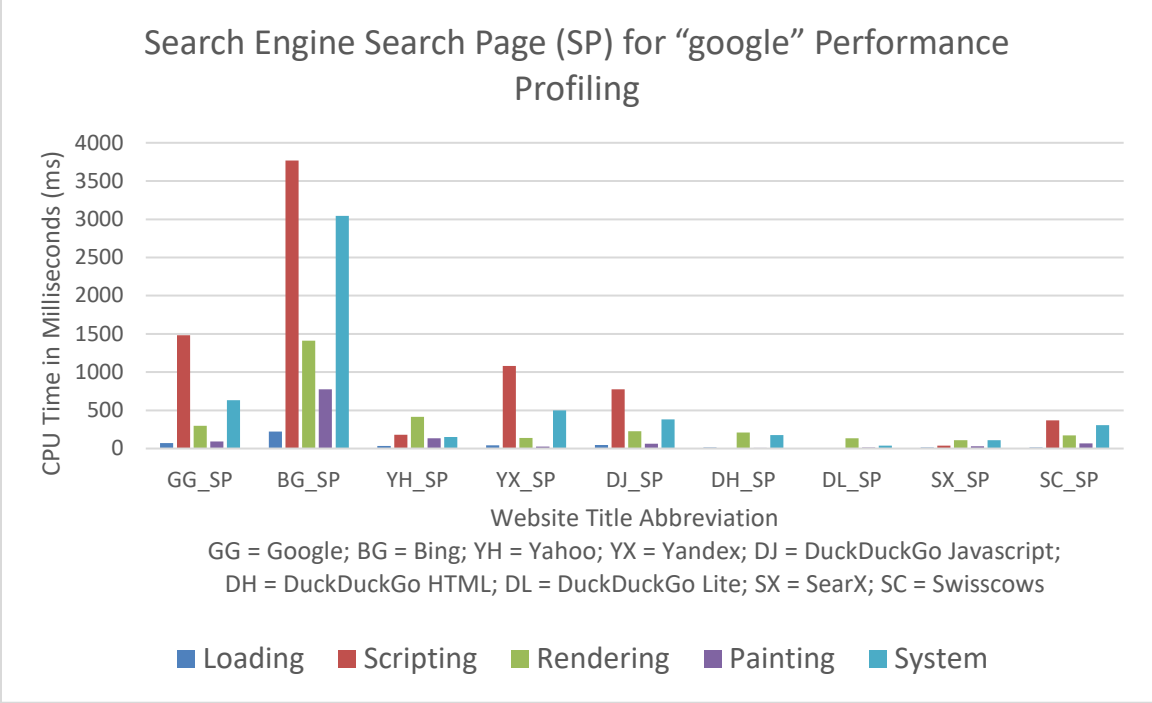

Figure 15: Second Collection Search Engine Search for “google” Performance Profiling.

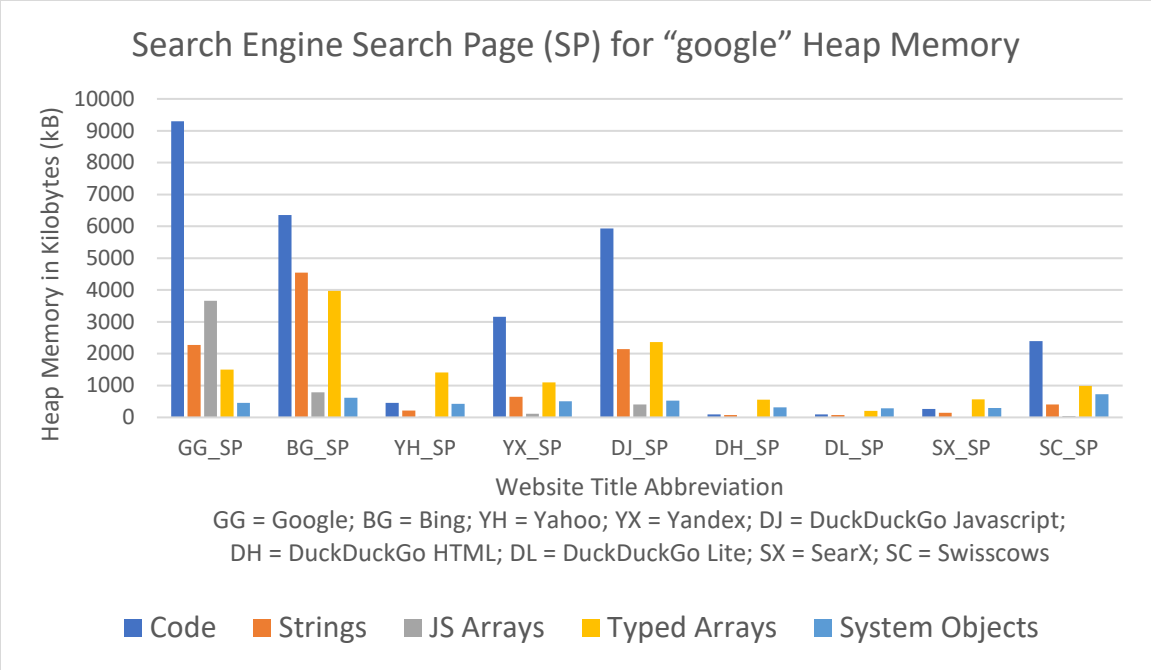

Figure 16: Second Collection Search Engine Search for “google” Heap Memory.

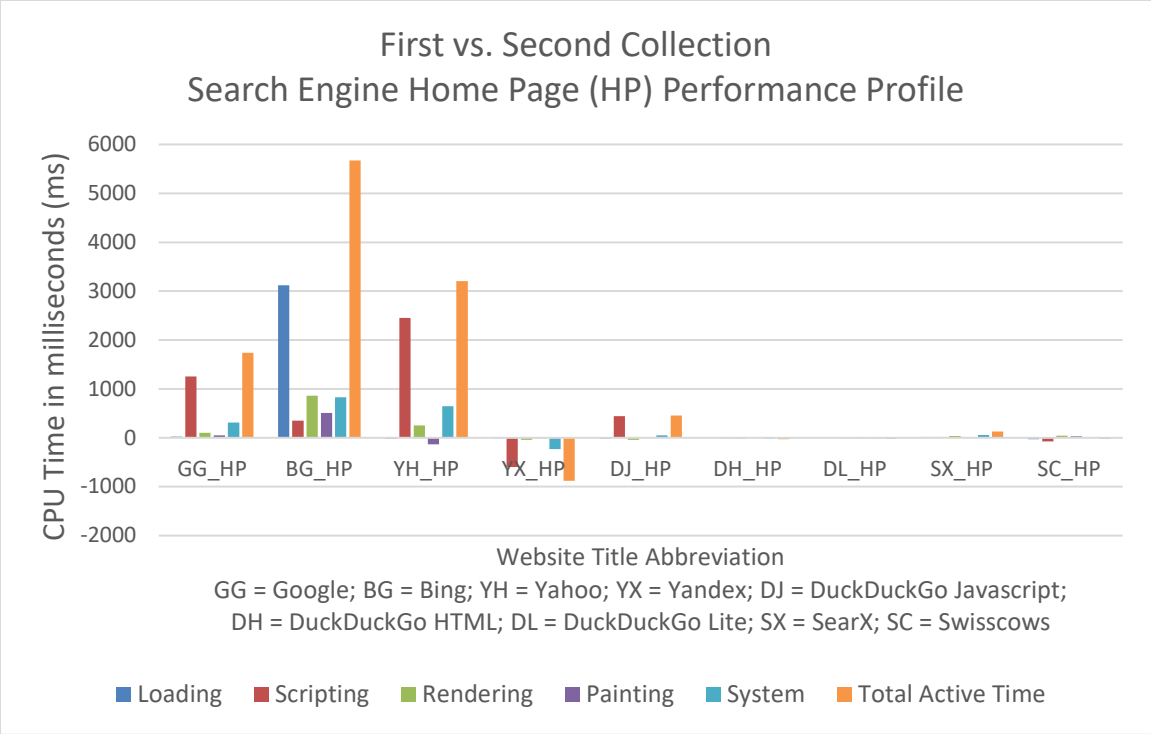

Figure 17: First vs. Second Collection Search Engine Home Page Performance Profile.

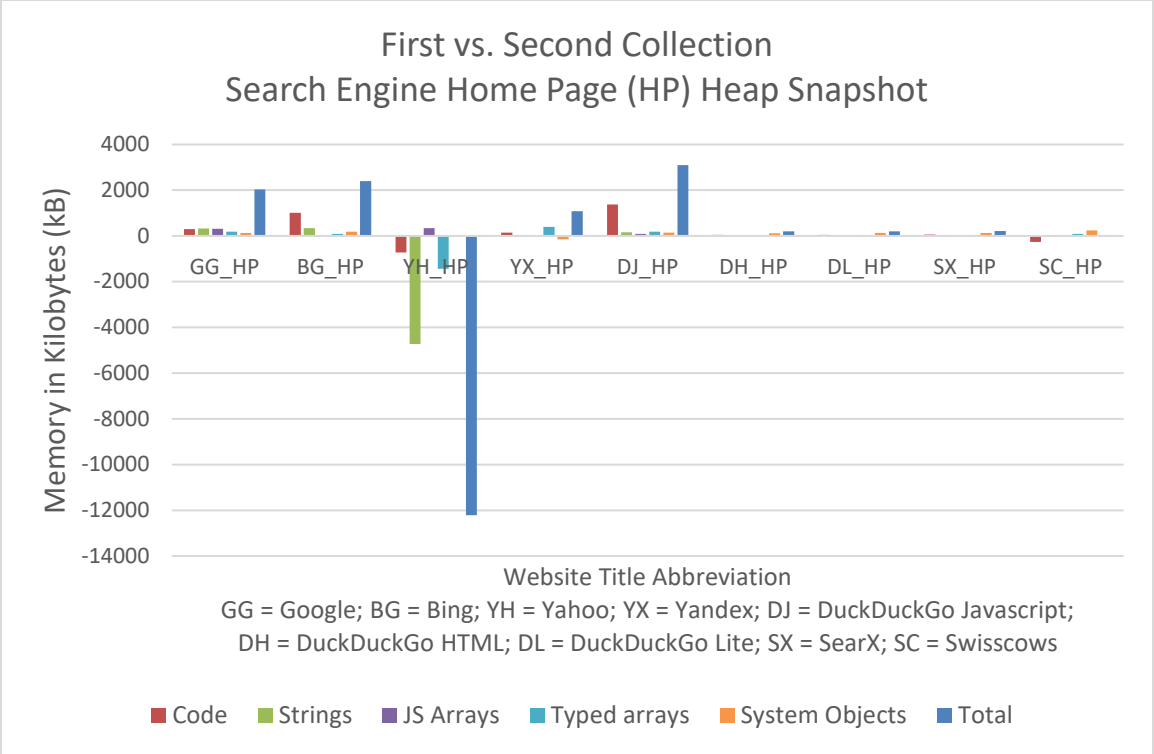

Figure 18: First vs. Second Collection Search Engine Home Page Heap Snapshot.

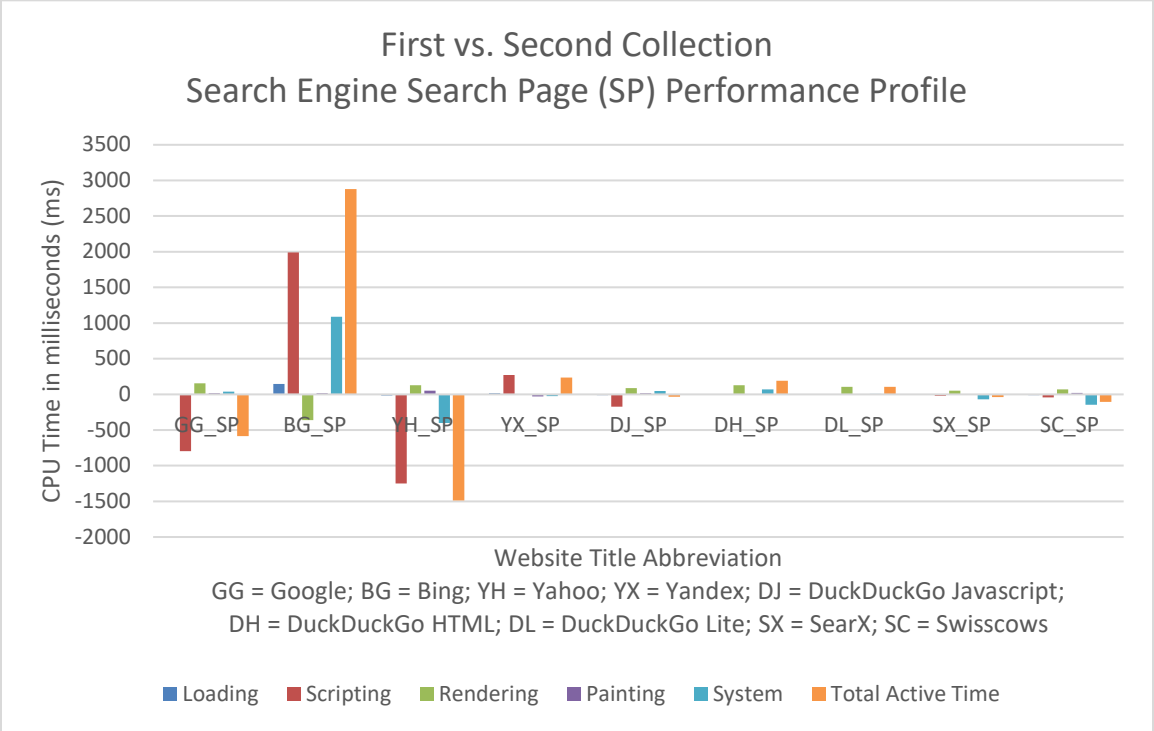

Figure 19: First vs. Second Collection Search Engine Search Page Performance Profile.

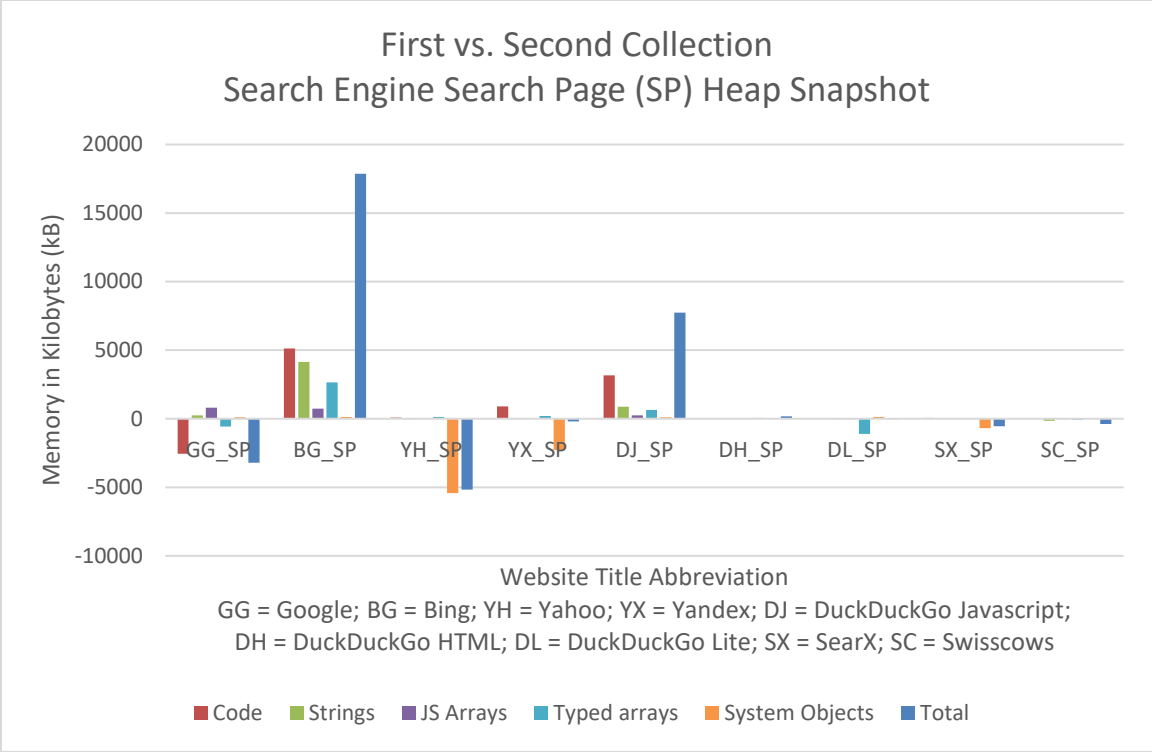

Figure 20: First vs. Second Collection Search Engine Search Page Heap Snapshot.

News Sites

First Collection. On the news sites scripting is the one thing that they do the most (Figure 21). Fox news is the highest in most categories but during testing CNN started playing live footage during profiling. This led to an automatic refresh, which could not be replicated because they mark your IP for no more live TV. The aggregate site is the winner here with the New York Times being the only non-aggregated site that comes close to it.

There is nothing much to differentiate the news sites when it comes to memory use (Figure 22). The only interesting finding is that CNN uses more memory for code, possibly because of its live TV streaming feature. This caused trouble in the initial testing because as profiling was happening the page reloaded because the live TV stream acted as a free trial. The website no longer automatically played the live TV stream as it

marked the IP address as having used up its free trial. Though this heap snapshot was after that happened, this could mean that CNN still loads in all the code to stream it before it checks trial status of the IP address.

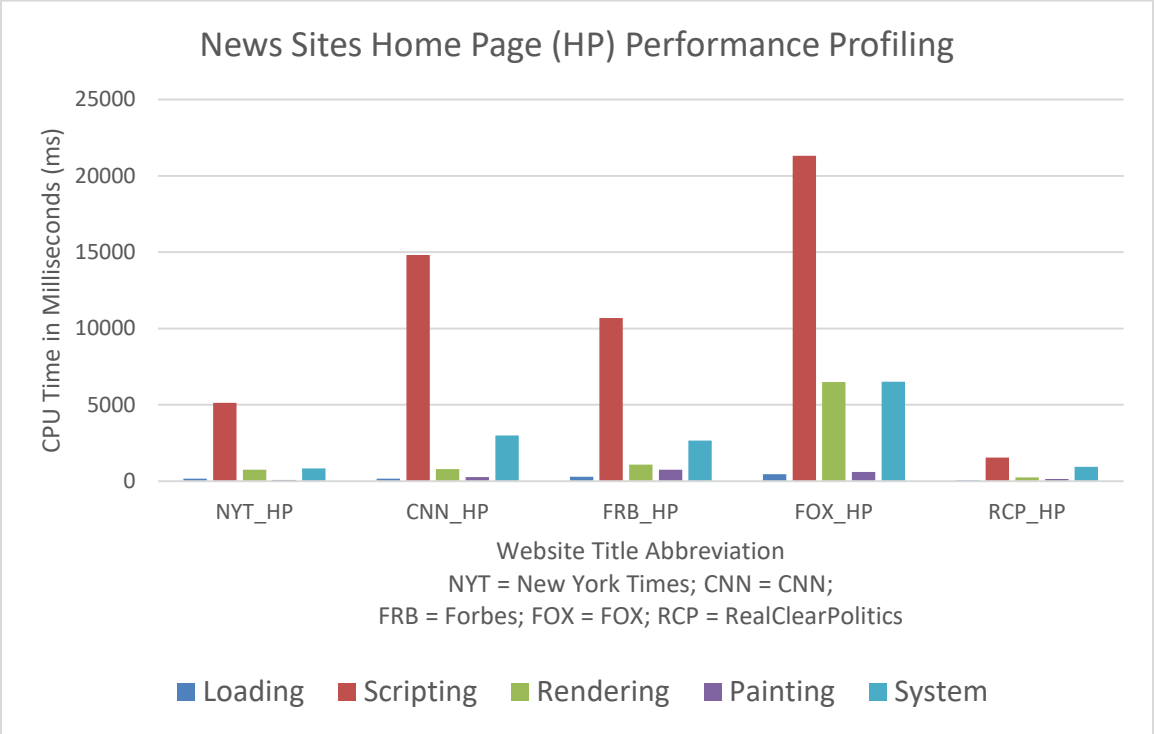

Figure 21: First Collection News Sites Home Page Performance Profiling.

Second Collection. For the new data there are few noticeable changes, primarily being CNN using a ton more resources than last time. It sees increases in its CPU time usage for both rendering and system (Figure 19/23). Though the most noticeable change is the increase from just under 15000 ms for scripting to over 80000 ms for the new data, over a five times increase. When profiling CNN it took over 100 seconds to fully load the page and complete the profile having heavy CPU usage all throughout the timeline. It could be possible that it was live streaming video and that is why so much scripting time was used but it was not visible on the screen when profiling. Interestingly for memory CNN has stayed overall the same with Forbes seeing increases in code and Fox also

seeing increases in code as well as strings (Figure 20/24).

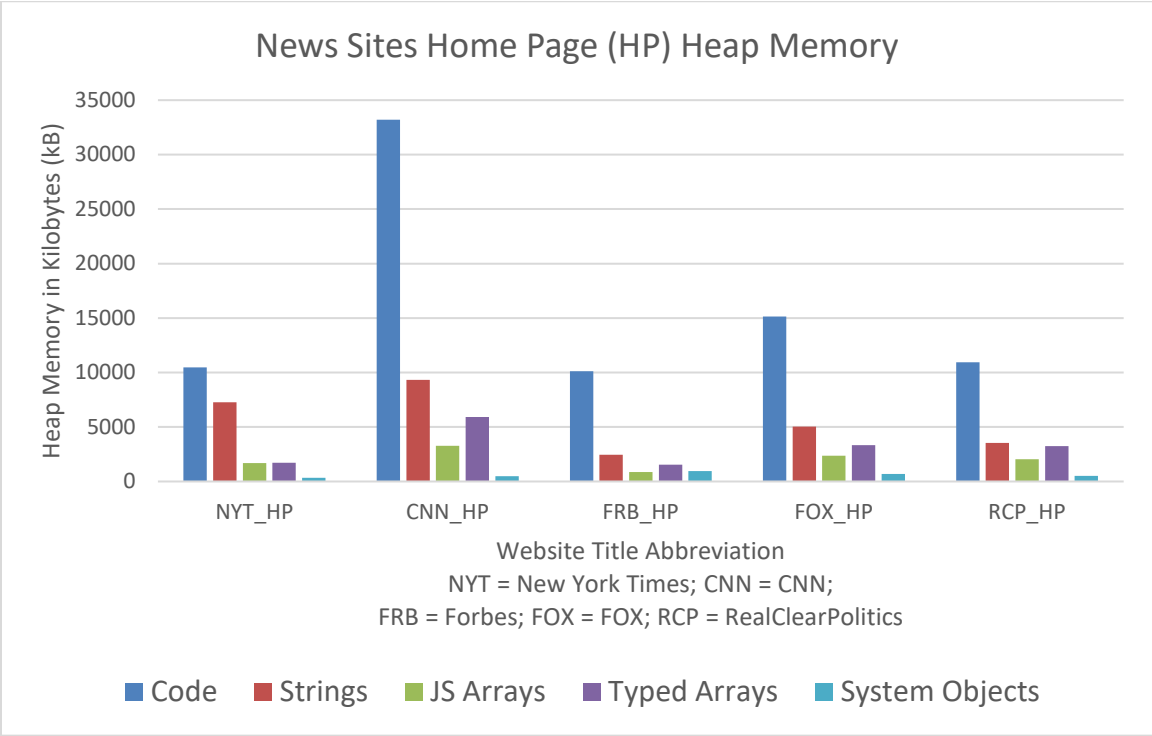

Figure 22: First Collection News Sites Home Page Heap Memory.

First vs. Second Collection. Figures 25 and 26 show the differences between the first and second collections for the new sites category indicating improvement or regression.

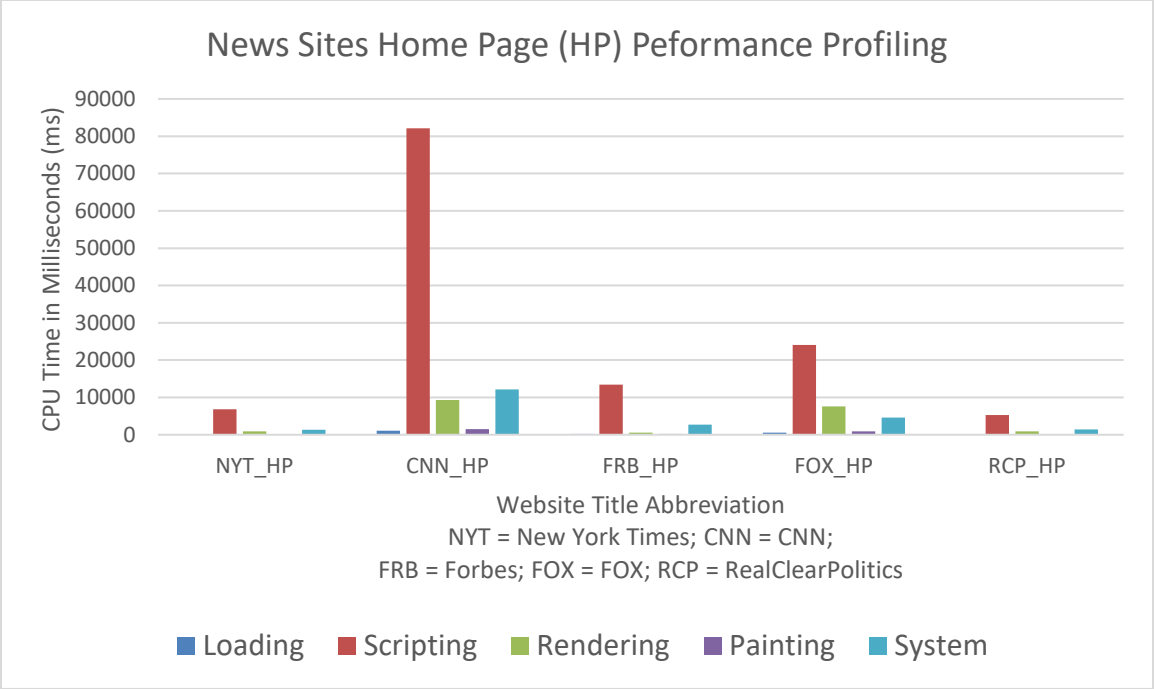

Figure 23: Second Collection News Sites Home Page Performance Profiling.

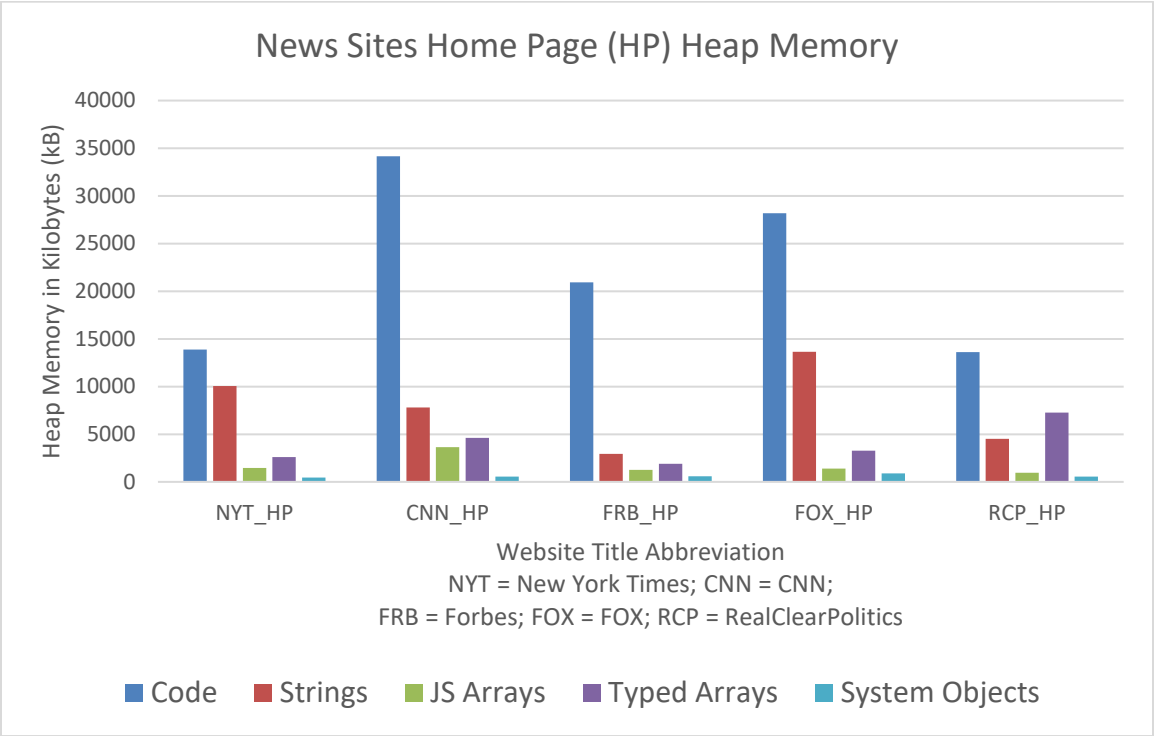

Figure 24: Second Collection News Sites Home Page Heap Memory.

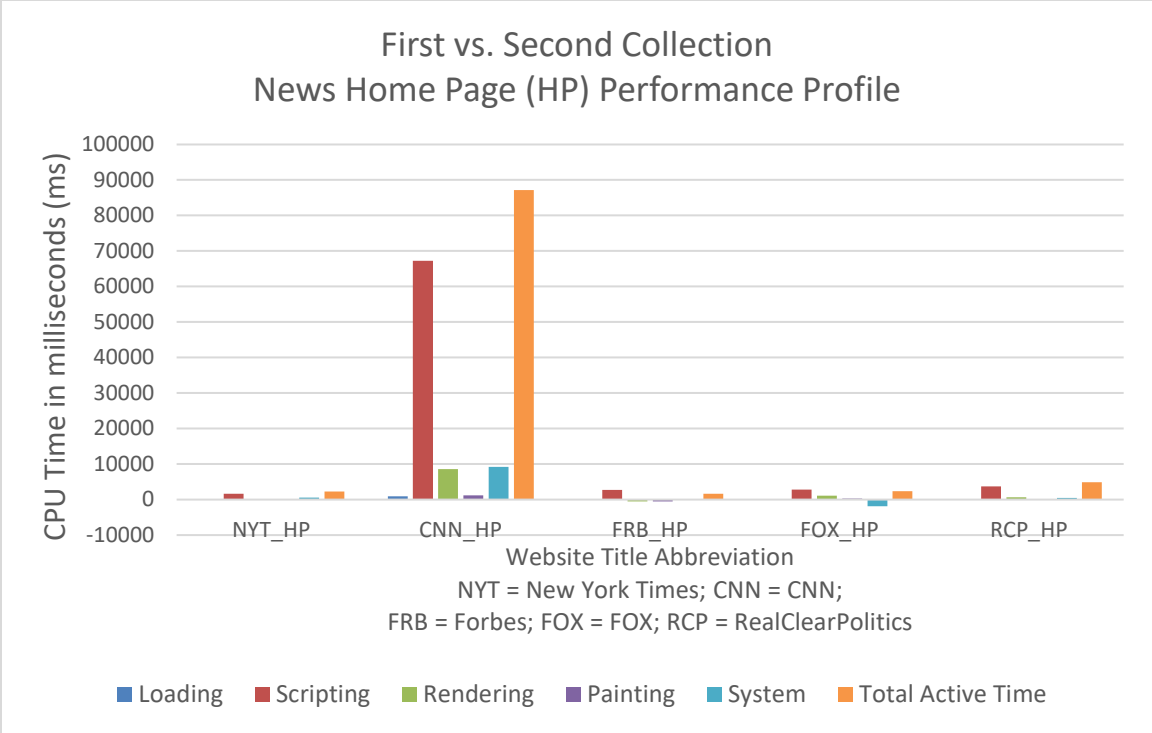

Figure 25: First vs. Second Collection News Home Page Performance Profile.

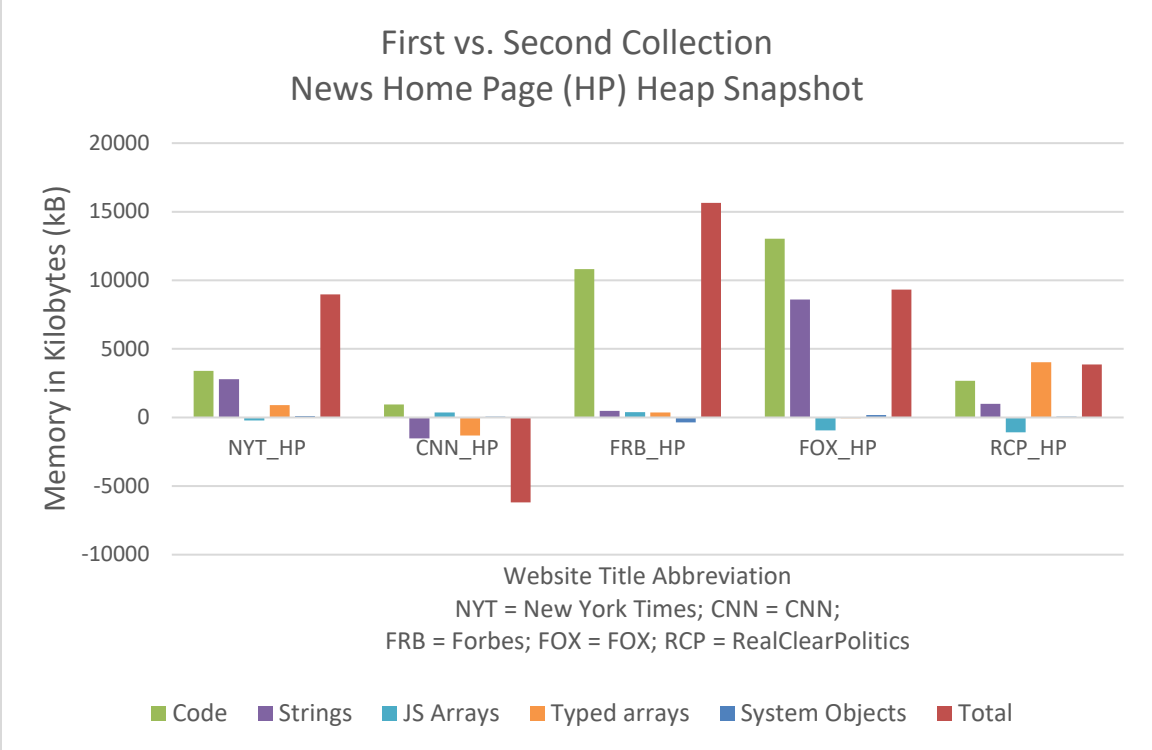

Figure 26: First vs. Second Collection News Home Page Heap Snapshot.

Online Shopping

First Collection. Online shopping is unfortunately not as interesting with all using most of the CPU time for scripting, a bit for system, and little for everything else (Figure 27). Wish uses the most CPU time and with the knowledge that Wish is a site that is sort of "sketchy," it might explain why.

Again, nothing interesting, all use a similar amount of memory of a similar configuration (Figure 28). Alibaba joins Wish for this one in using the most memory. Wish's distribution for its memory usage is different from all the others as the next biggest aspect taking up memory is strings and not typed arrays.

Second Collection. The only change for the new data is an increase all around for scripting while everything else stays about the same (Figure 29). Memory for online shopping follows the same patterns of allocation for each website but sees overall increases for Etsy, Walmart, Alibaba, and Wish, while Amazon as well as eBay see an overall decrease (Figure 30).

First vs. Second Collection. Figures 31 and 32 show the differences between the first and second collections for the online shopping category indicating improvement or regression.

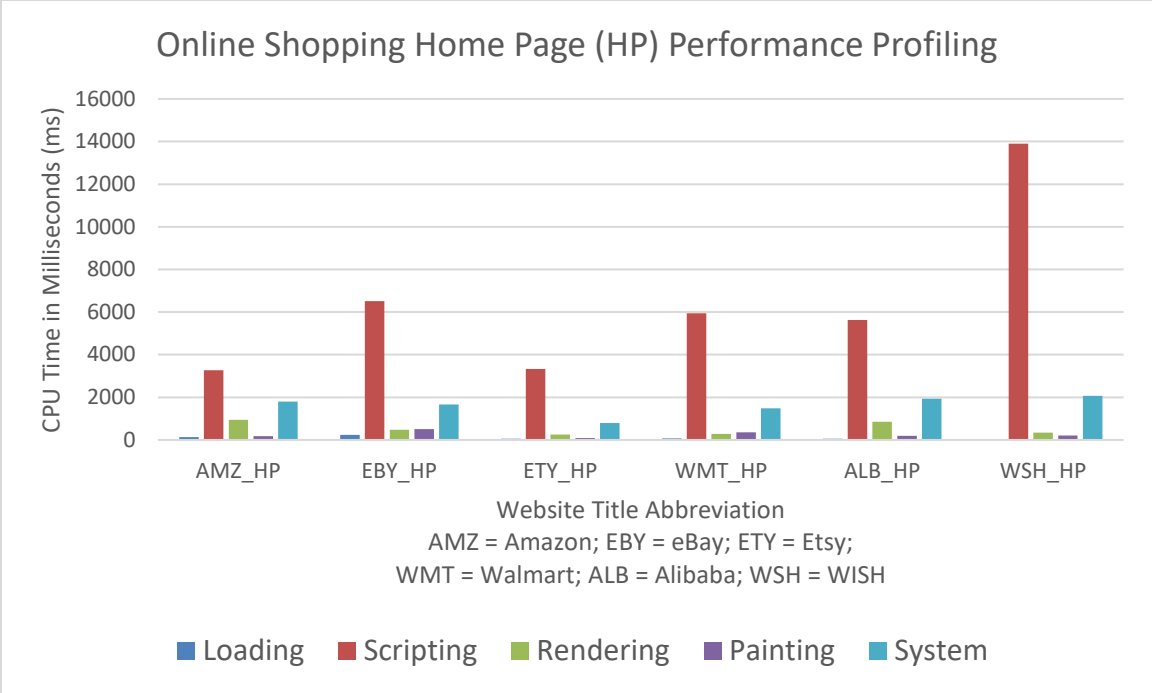

Figure 27: First Collection Online Shopping Home Page Performance Profiling.

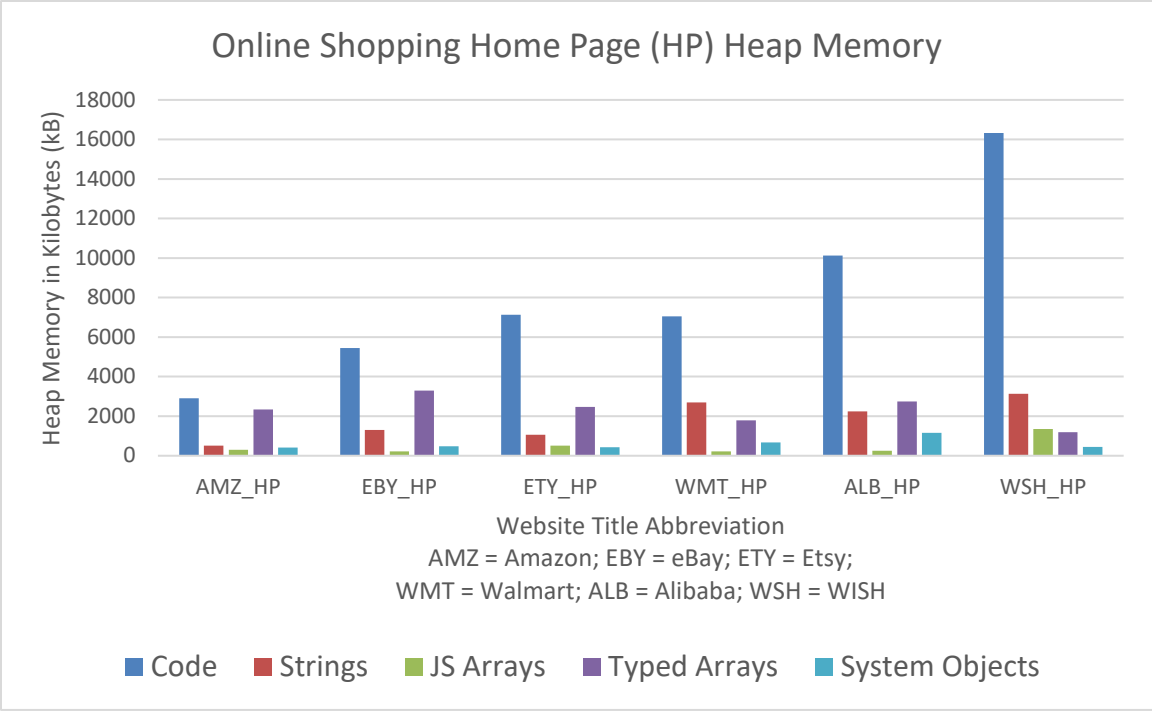

Figure 28: First Collection Online Shopping Home Page Heap Memory.

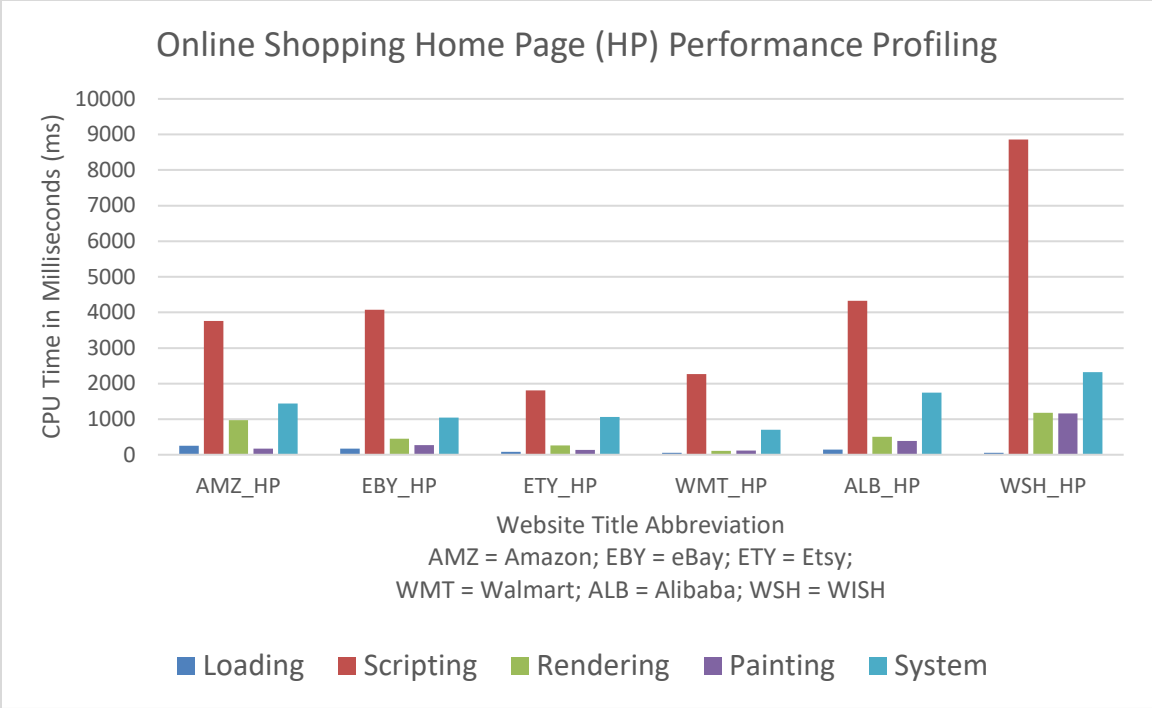

Figure 29: Second Collection Online Shopping Home Page Performance Profiling.

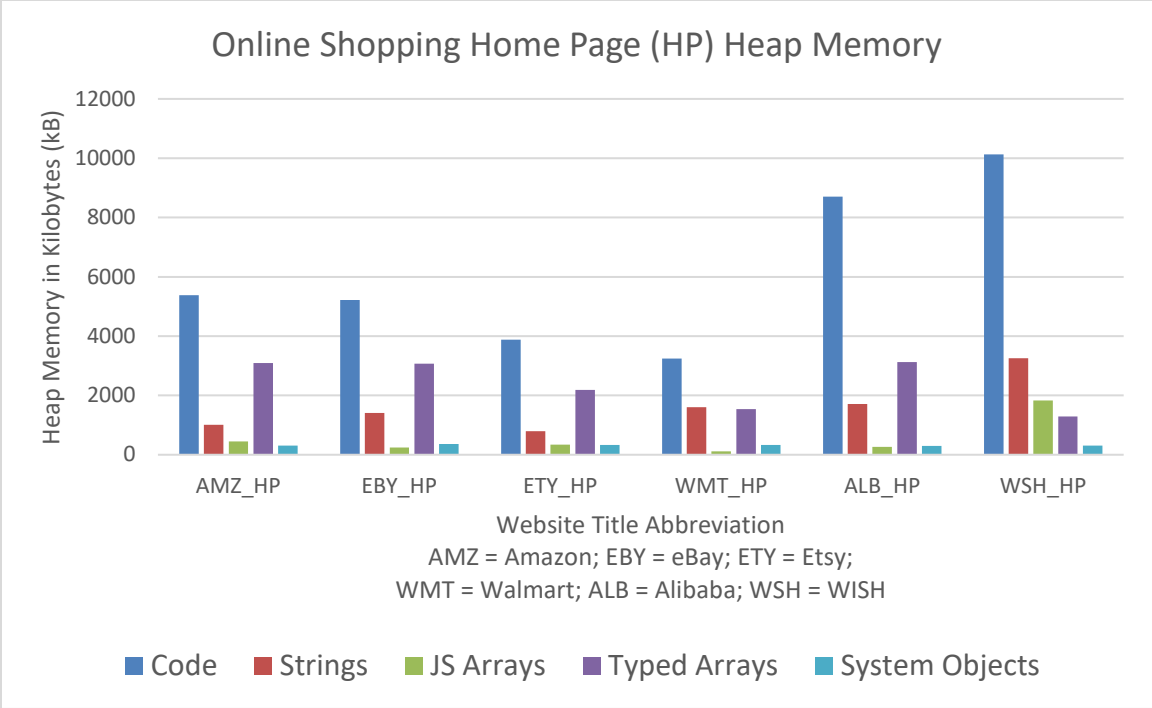

Figure 30: Second Collection Online Shopping Home Page Heap Memory.

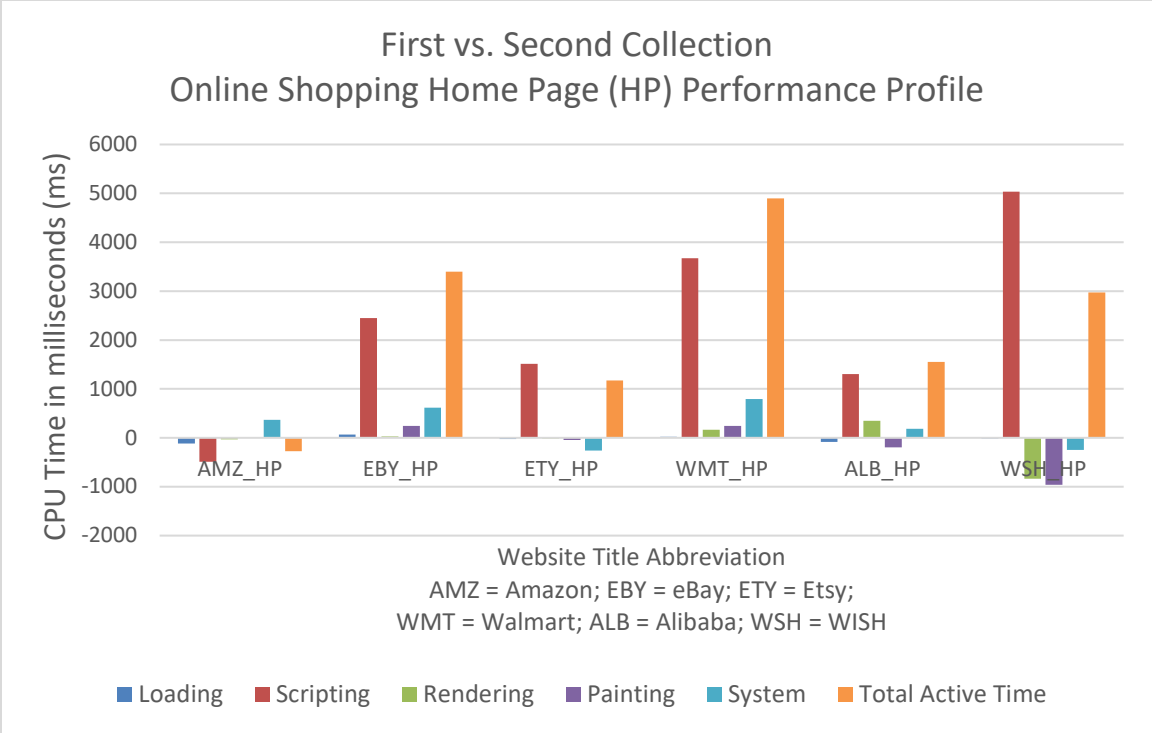

Figure 31: First vs. Second Collection Online Shopping Home Page Performance Profile.

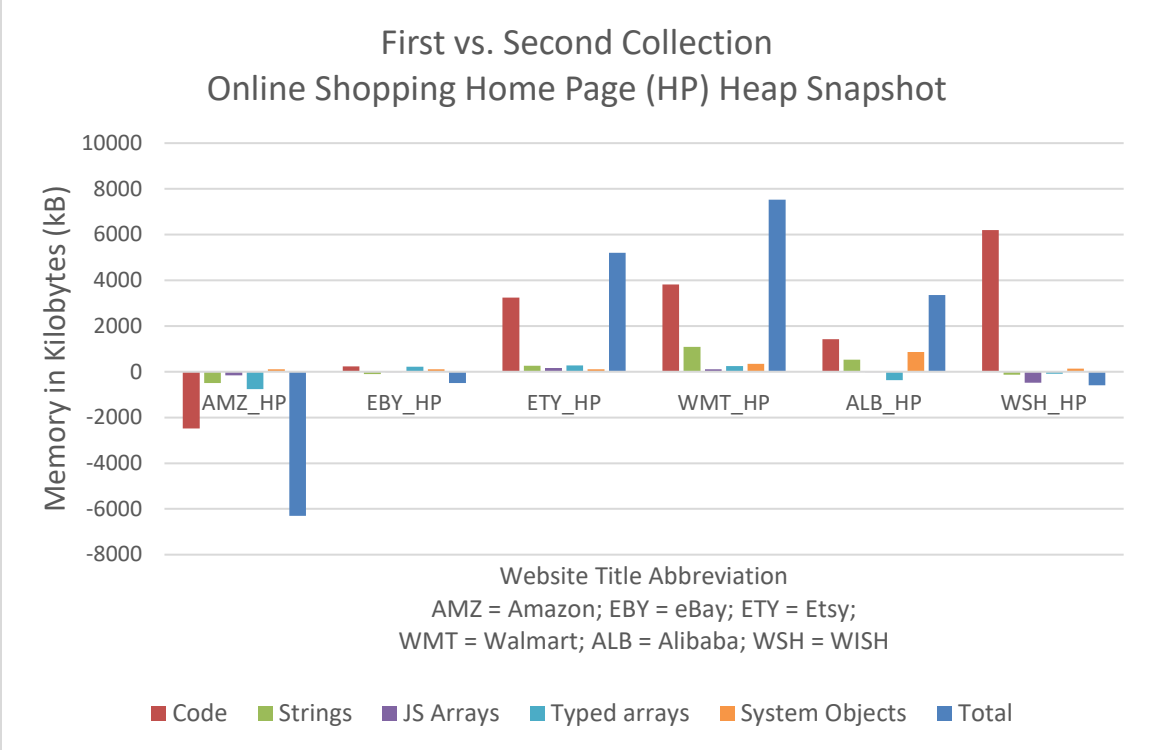

Figure 32: First vs. Second Collection Online Shopping Home Page Heap Snapshot.

Social Media

First Collection. Twitter seems to either have been preloaded to Google Chrome not allowing the profiler to get numbers for profiling or does just not work well with it (Figure 33). Interestingly Getter, an Alt-Tech site, and Reddit, a Big Tech site both use a large amount of CPU time. The previous version of reddit uses a fraction of the CPU time at the cost of looking "nice" but that can be all a matter of personal preference. Old Reddit and 4chan are similar in CPU time usage but differ in that Old Reddit's is more spread out while 4chan has it all in scripting.

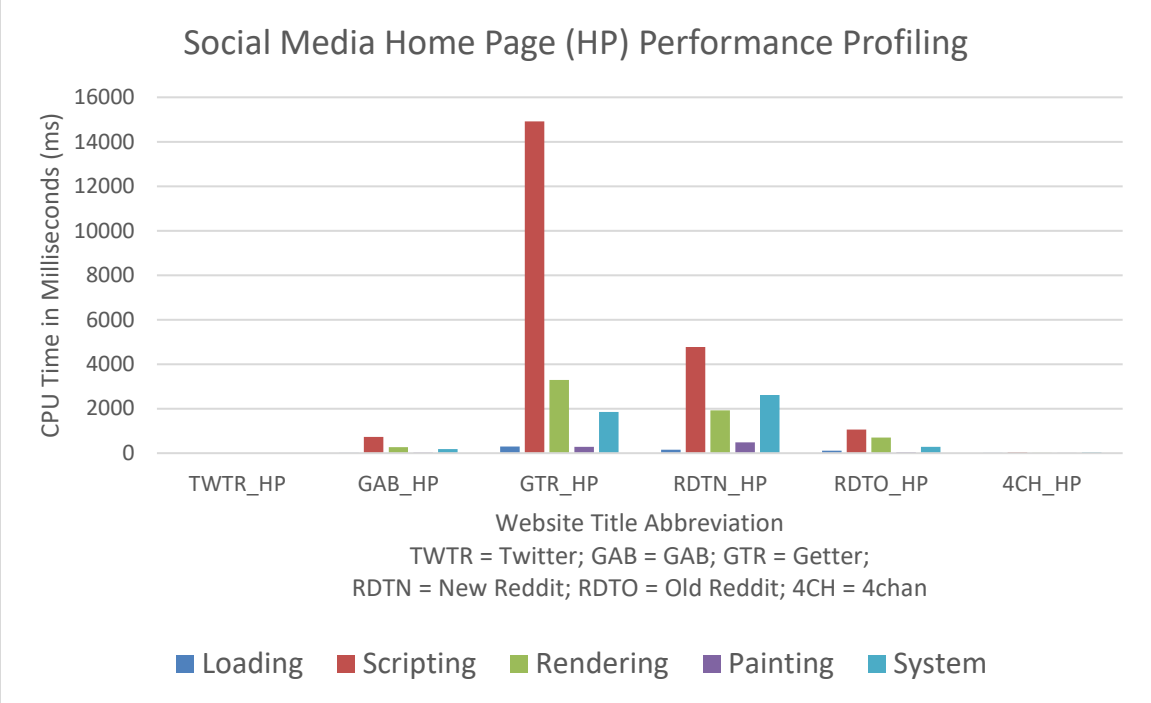

Figure 33: First Collection Social Media Home Page Performance Profiling.

Getter dwarfs the rest of the sites through memory for system objects (Figure 34). Twitter and New Reddit are the only ones that use up a noticeable amount of memory while the rest of the sites can be forgotten about if trying to free up memory.

Second Collection. Unfortunately, for the new data, Twitter, now named X, is

unavailable due to the requirement to login to view the site except if you have a link to a specific tweet and even then, the comments for it cannot be seen (Figure 35). There were also problems with profiling 4chan without it being precached by the browser. On first load it takes about ten or more captcha verifications for the page to finally load. Profiling when precached and comparing it to the previous data shows a decrease in CPU time, especially in scripting.

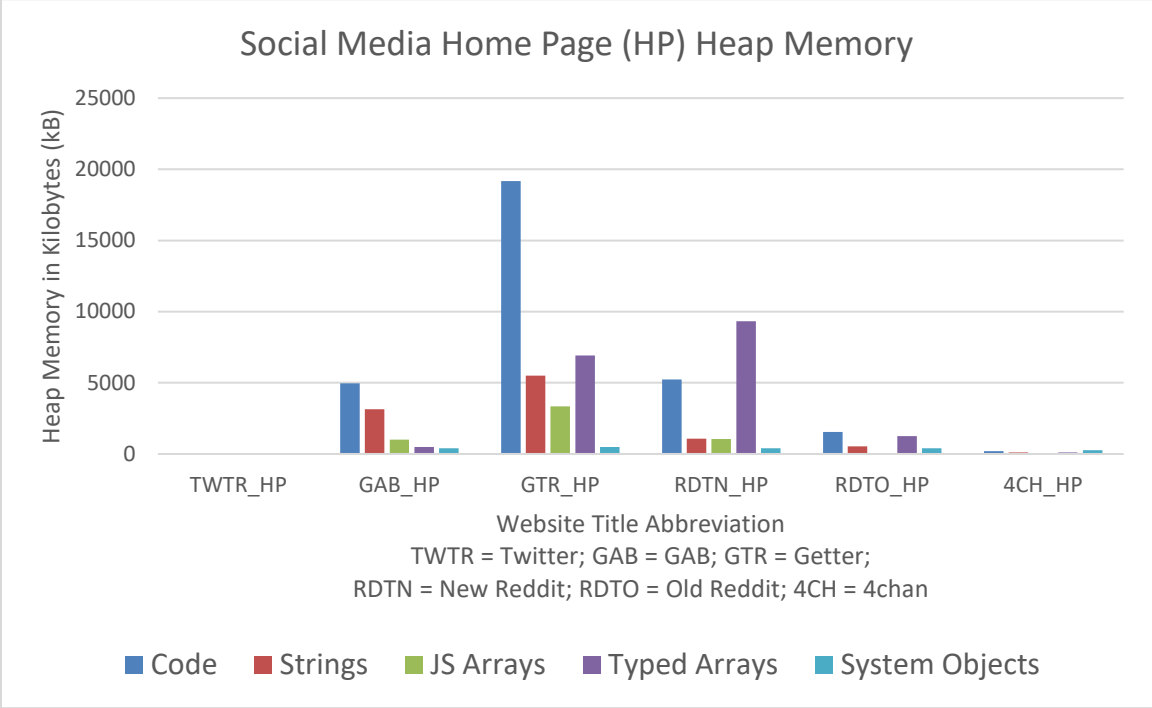

Figure 34: First Collection Social Media Home Page Heap Memory.

The data that was able to be profiled reliably shows Gab having a decrease in every task along with Old Reddit. Getter has seen an increase in CPU time usage from an already resource hungry site. New Reddit seems its most intensive tasks going from being just scripting to that being cut in half and seeing increases in rendering, painting, and system. New social media memory brings good news as Getters and New Reddit's memory are more than or nearly halved, Gab also sees a small decrease.

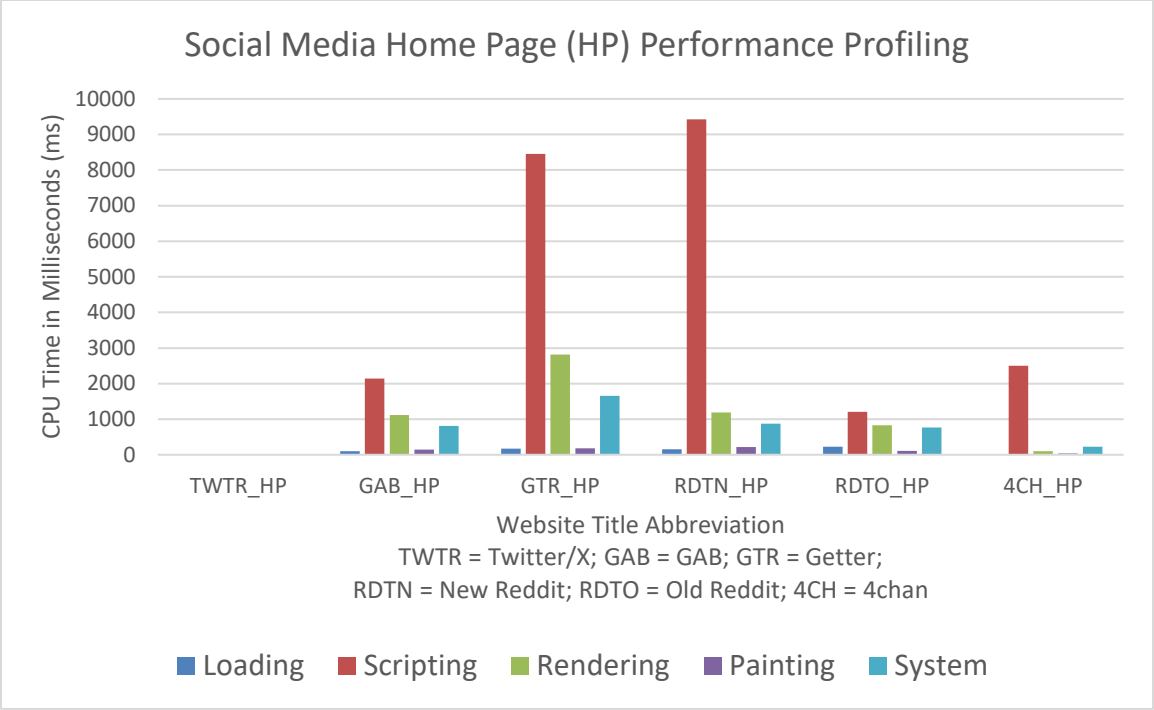

Figure 35: Second Collection Social Media Home Page Performance Profiling.

Despite not being able to get an accurate CPU profile for 4chan, the memory profile will be unaffected by caching. So, it still being the most memory efficient is still relevant (Figure 36).

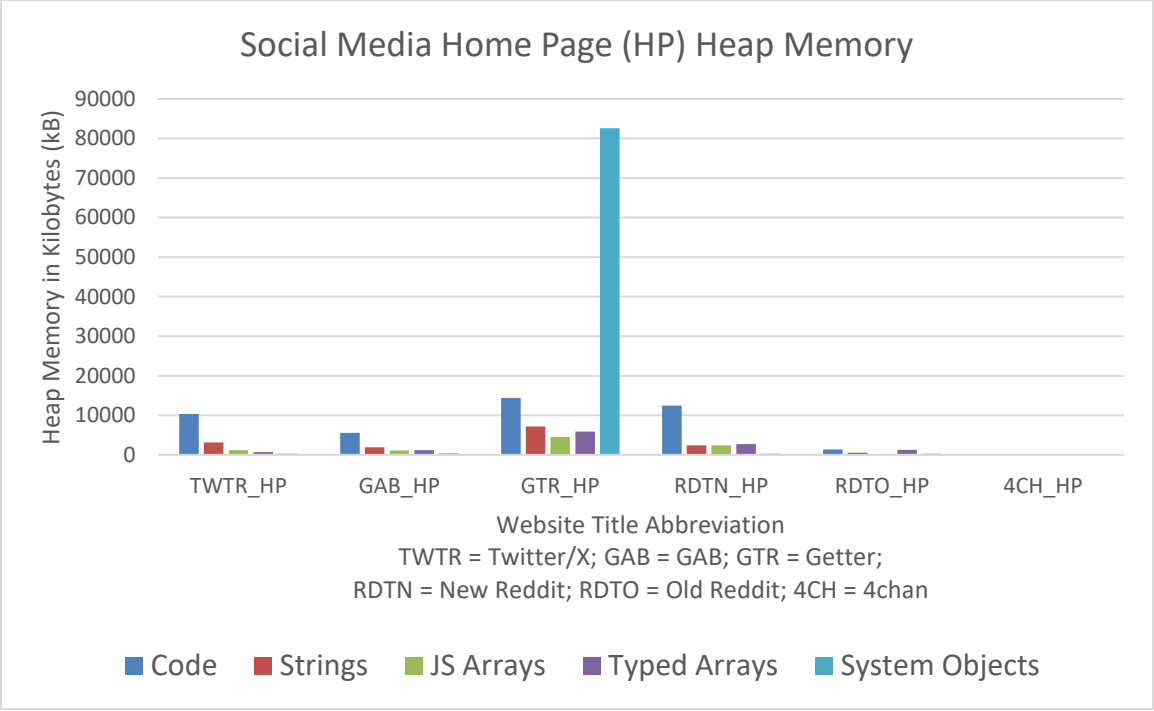

Figure 36: Second Collection Social Media Home Page Heap Memory.

First vs. Second Collection. Figures 37 and 38 show the differences between the first and second collections for the search engine category indicating improvement or regression.

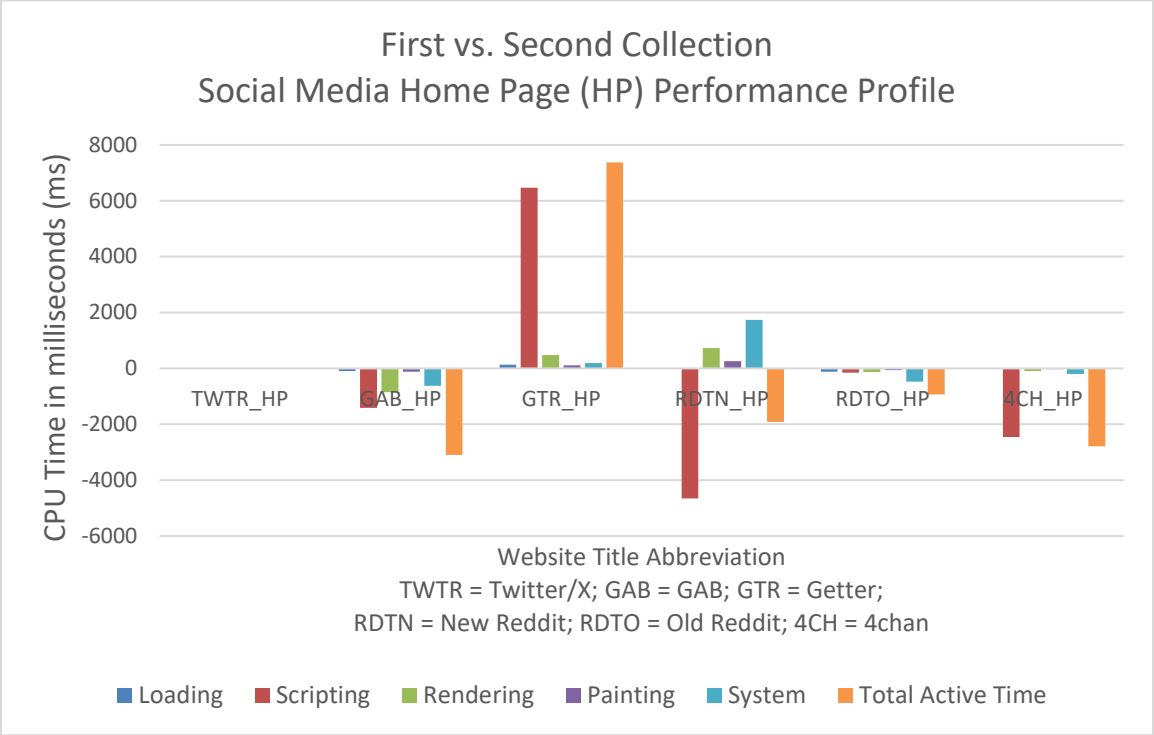

Figure 37: First vs. Second Collection Social Media Home Page Performance Profile.

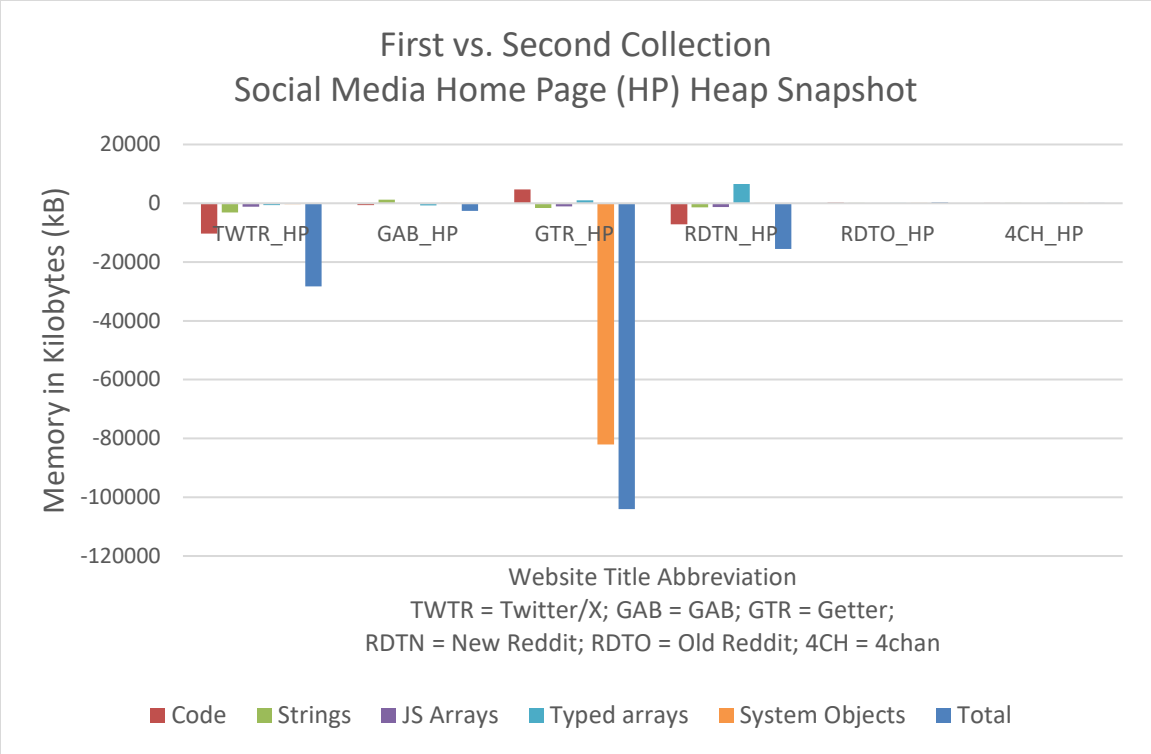

Figure 38: First vs. Second Collection Social Media Home Page Heap Snapshot.

Video Hosting

First Collection. YouTube uses a great amount of CPU time scripting followed by Odysee and then Dailymotion (Figure 39). BitChute and Rumble are the lowest and do not use much of anything.

Story is similar for memory YouTube uses an equal amount of memory for code and typed arrays, Odysee uses a ton of strings and half of that space for code, and Dailymotion puts it all into code (Figure 40). BitChute and Rumble are very resource efficient and do not use a lot of memory.

Once video playback begins YouTube uses the most CPU time and uses much of that scripting (Figure 41). Odysee and Dailymotion use an amount of CPU time right in the middle between YouTube and then at the lower end BitChute and Rumble. It should be noted though that profiling was done with the auto resolution, which is default for all

sites except BitChute which does not have a quality setting. YouTube auto selected to 720p while the rest were around 480p.

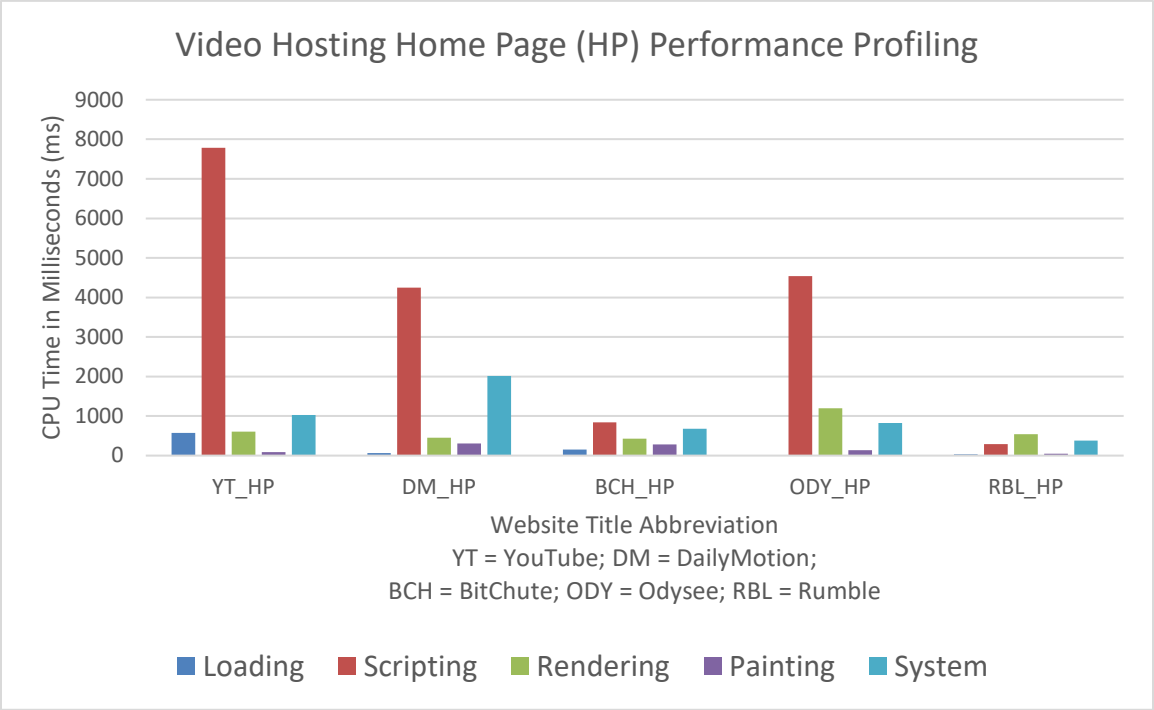

Figure 39: First Collection Video Hosting Home Page Performance Profiling.

YouTube, Odysee, and Dailymotion continue to be the largest resource users for video (Figure 42). The same pattern appears again for the memory as it did for the home pages. Unfortunately, the allocation timeline was of no use in comparing these sites because the video was so short, under thirty seconds, that all the data the needed to be downloaded was done all at once.

Second Collection. The video hosting sites homes pages all seem to be using less CPU time (Figure 43). Especially for YouTube since instead of the home page displaying videos it prompts you to search instead.

Memory usage is greater for all video hosting sites except YouTube most likely due to the previously mentioned sanitized home page (Figure 44).

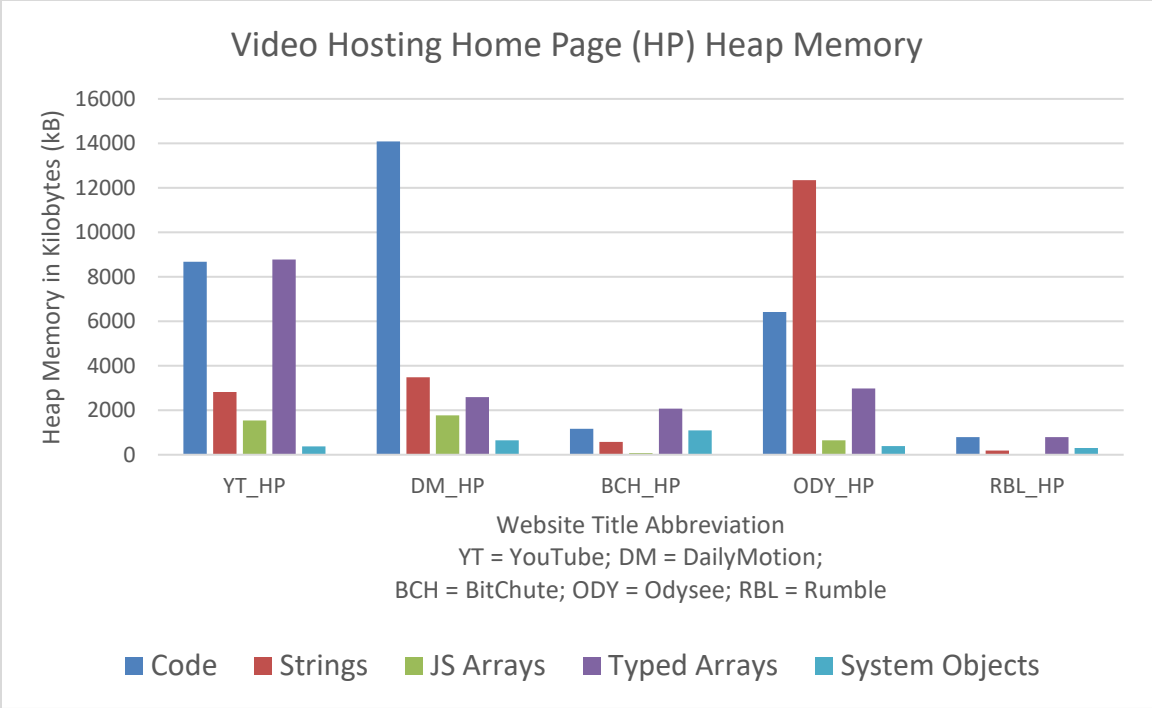

Figure 40: First Collection Video Hosting Home Page Heap Memory.

Video playback profiling of the CPU for the new data reveals that it is nearly the same as the old data (Figure 45). YouTube has seen a cut to its scripting while BitChute has interestingly seen an increase.

Just like the CPU data the Memory data is nearly identical as well, YouTube, BitChute, and Rumble do see a small increase in memory usage primarily in code though (Figure 46).

First vs. Second Collection. Figures 47, 48, 49, and 50 and show the differences between the first and second collections for the search engine category indicating improvement or regression.

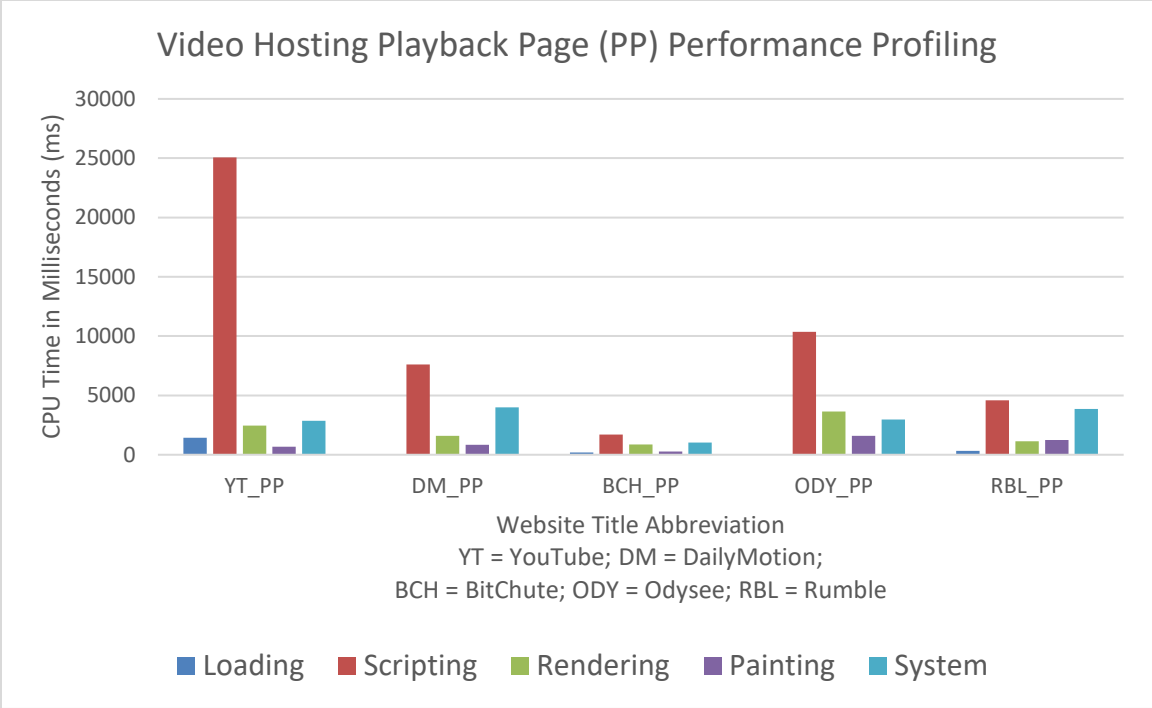

Figure 41: First Collection Video Hosting Playback Page Performance Profiling.

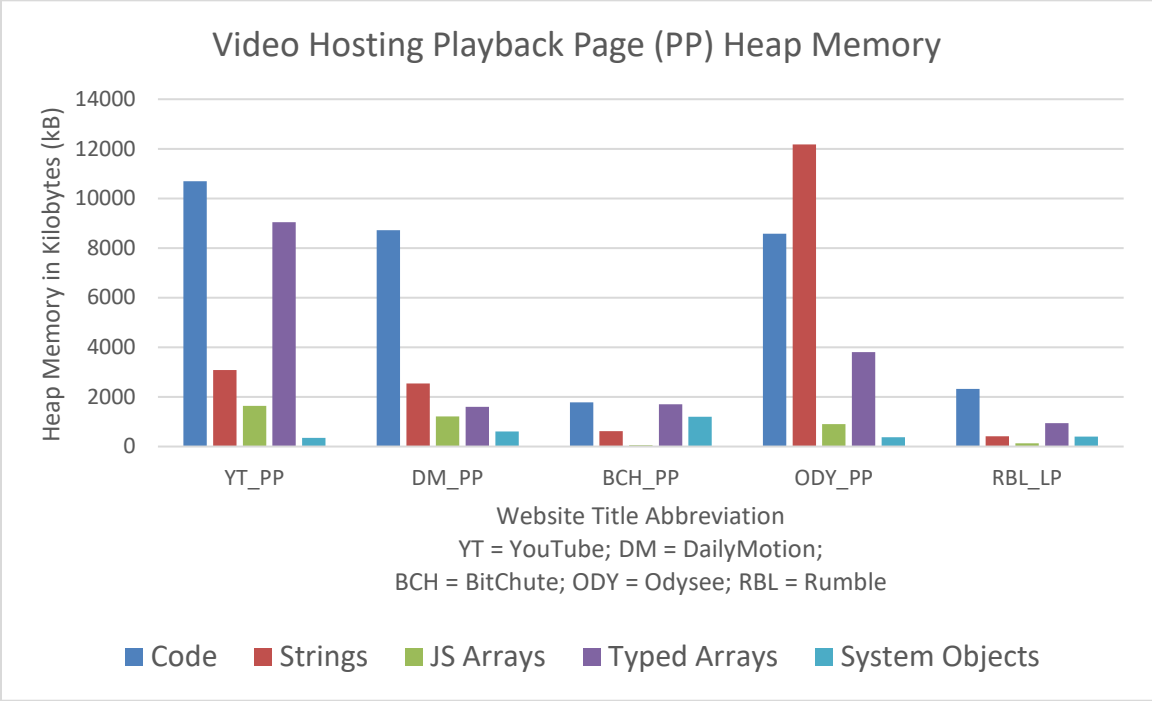

Figure 42: First Collection Video Hosting Playback Page Heap Memory.

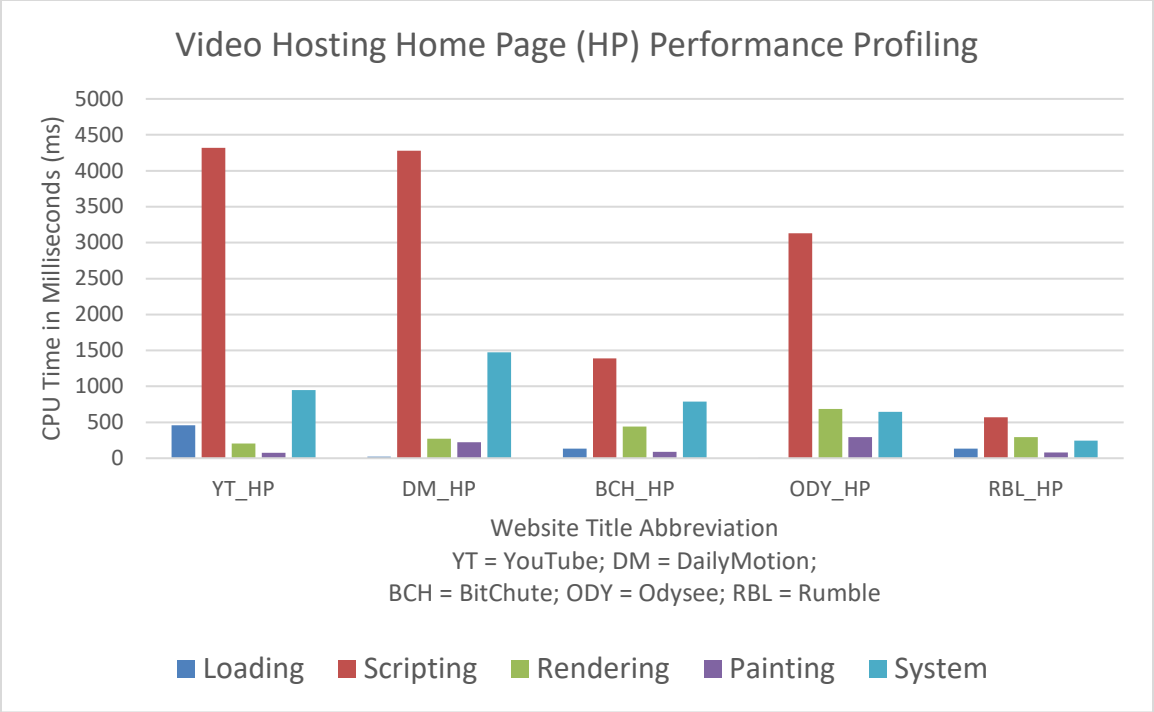

Figure 43: Second Collection Video Hosting Home Page Performance Profiling.

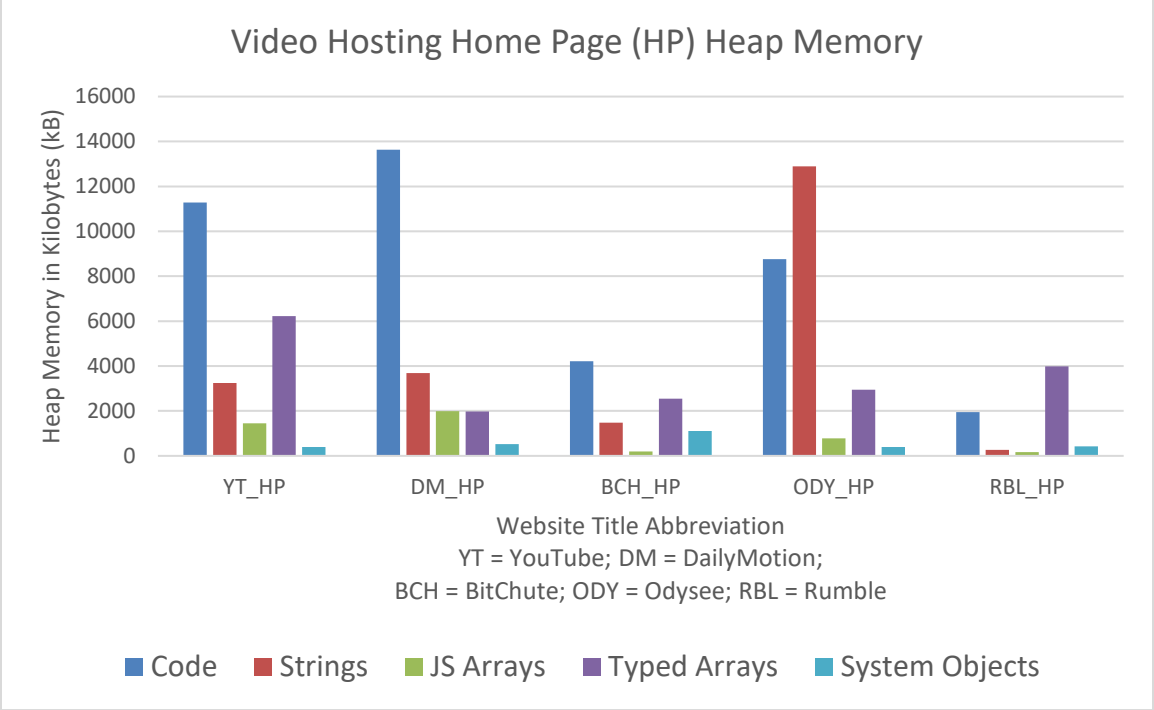

Figure 44: Second Collection Video Hosting Home Page Heap Memory.

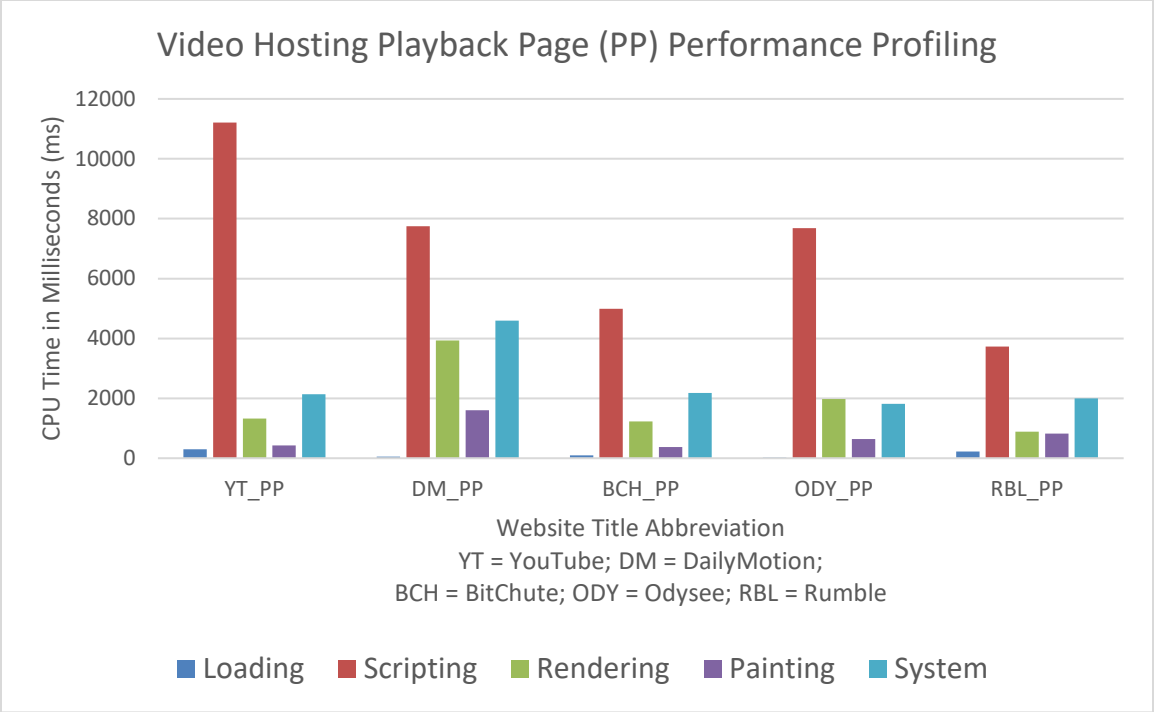

Figure 45: Second Collection Video Hosting Playback Page Performance Profiling.

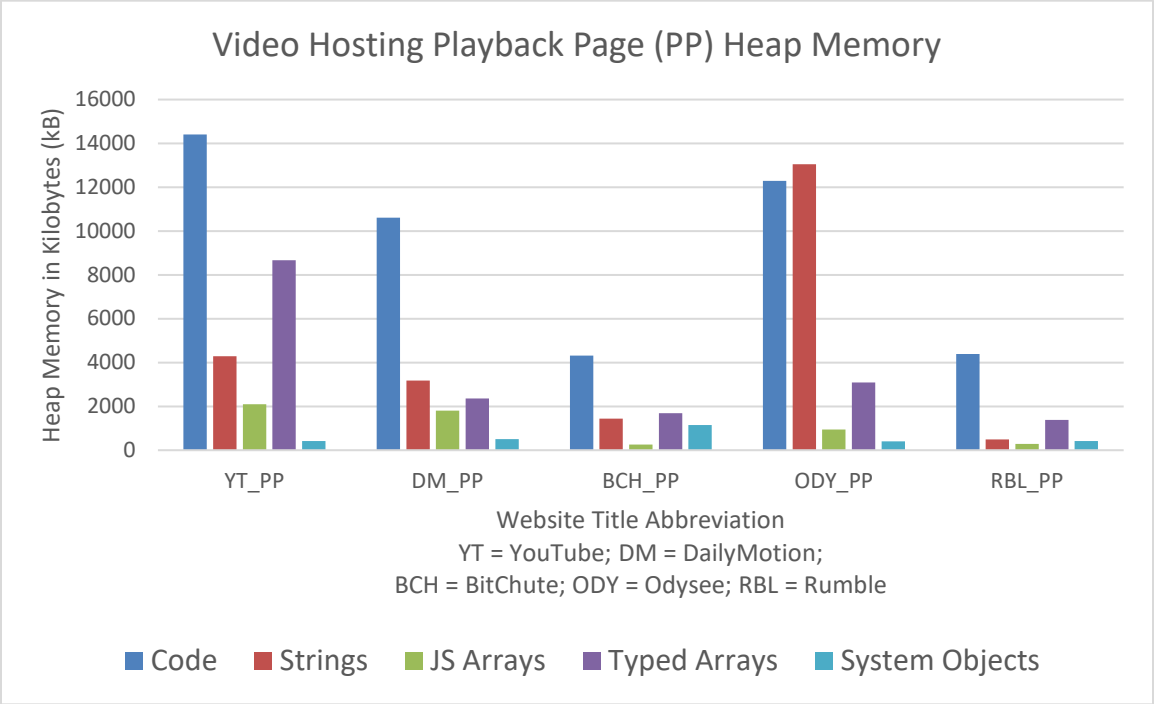

Figure 46: Second Collection Video Hosting Playback Page Heap Memory.

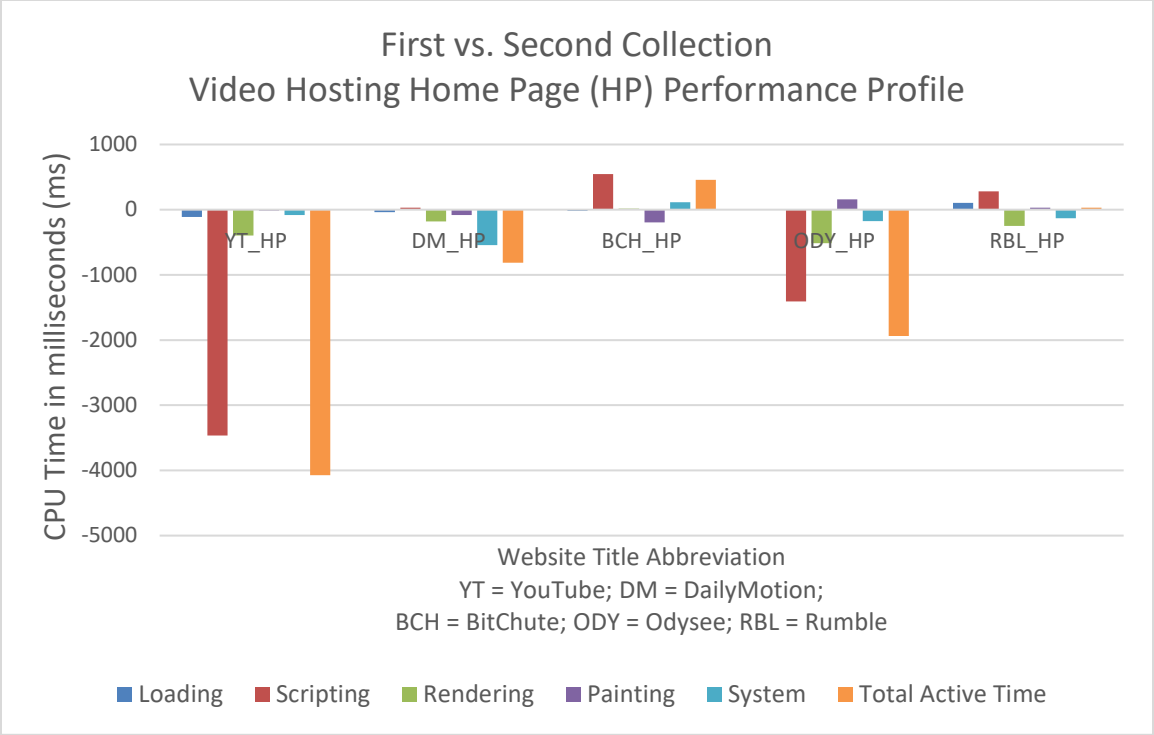

Figure 47: First vs. Second Collection Video Hosting Home Page Performance Profile.

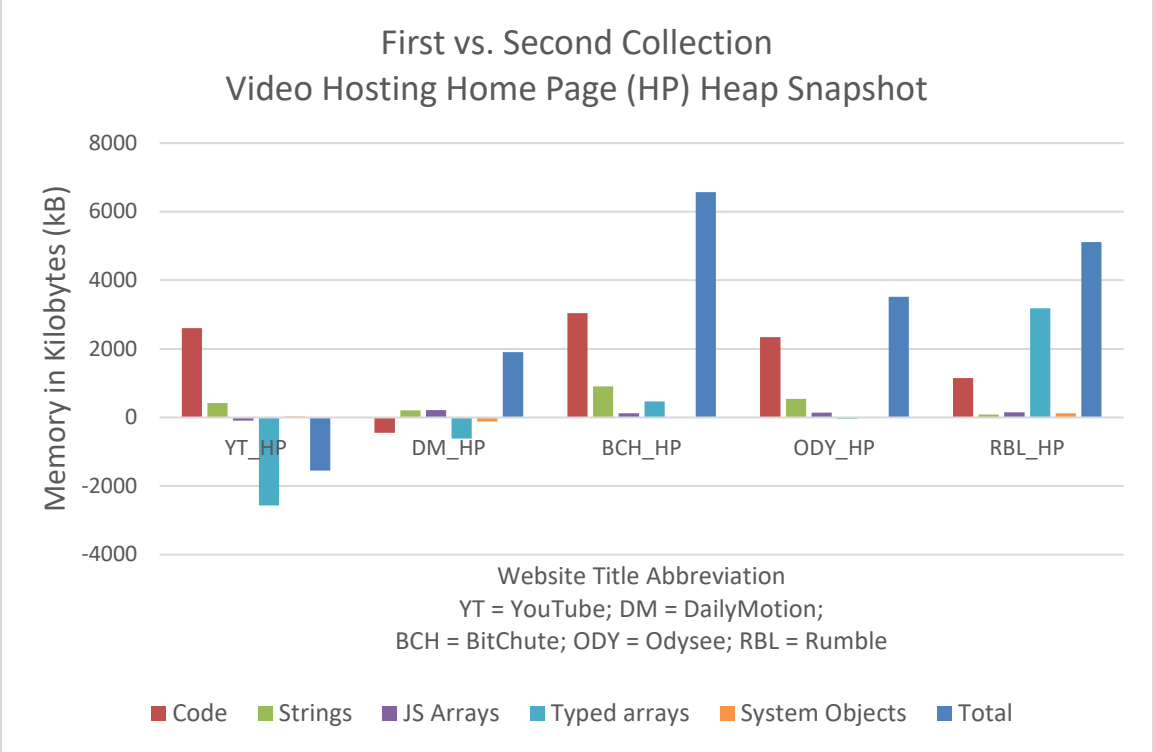

Figure 48: First vs. Second Collection Video Hosting Home Page Heap Snapshot.

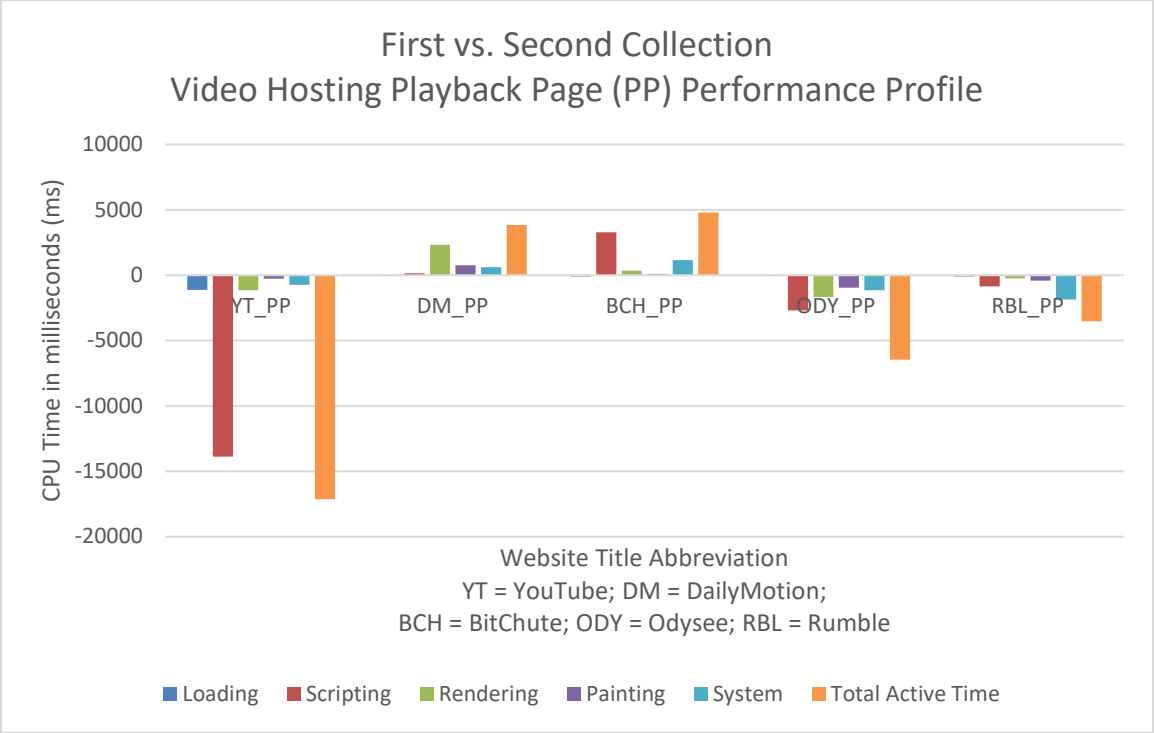

Figure 49: First vs. Second Collection Video Hosting Playback Page Performance Profile.

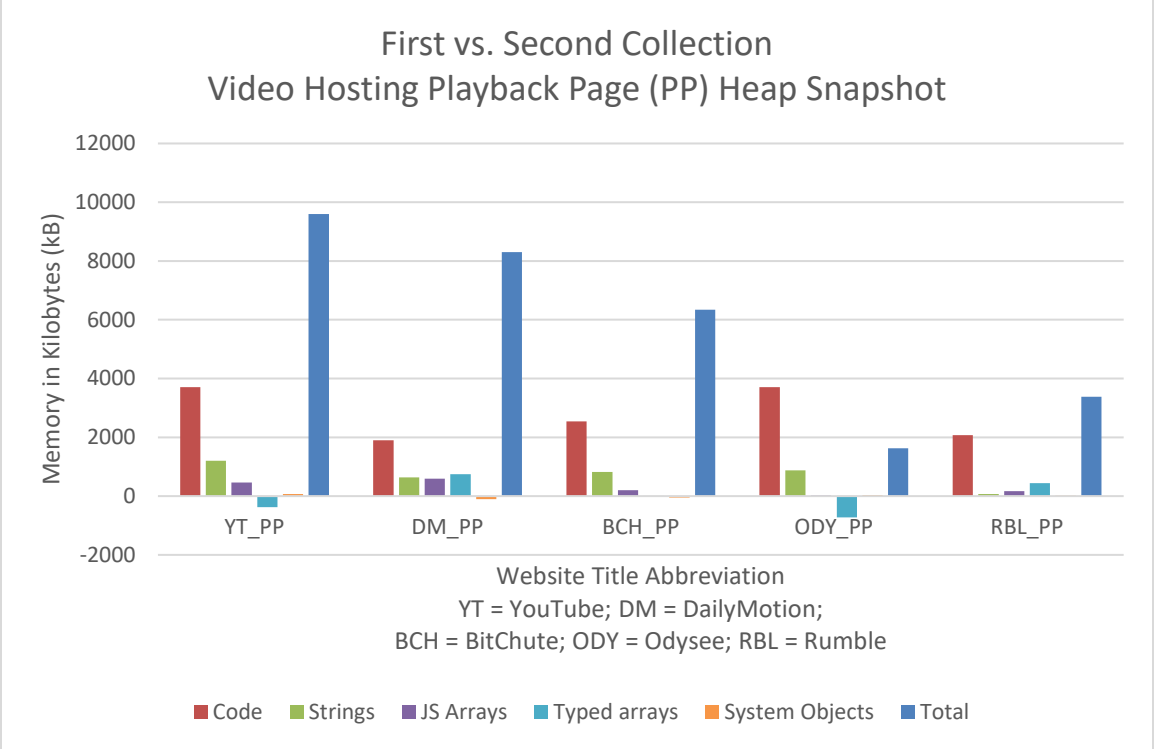

Figure 50: First vs. Second Collection Video Hosting Playback Page Heap Snapshot.

Web Apps

Web apps profiled using Chrome built in one show relatively little CPU time usage for the office suite compared to Ookla, a single suppose site that ends up using more CPU time both still and active because of ads (Figure 51).

The pattern of Ookla towering over the office suite in resource use continues into memory (Figure 52). The skew of the data in memory, and also in CPU time, is the same between the three office suite applications.

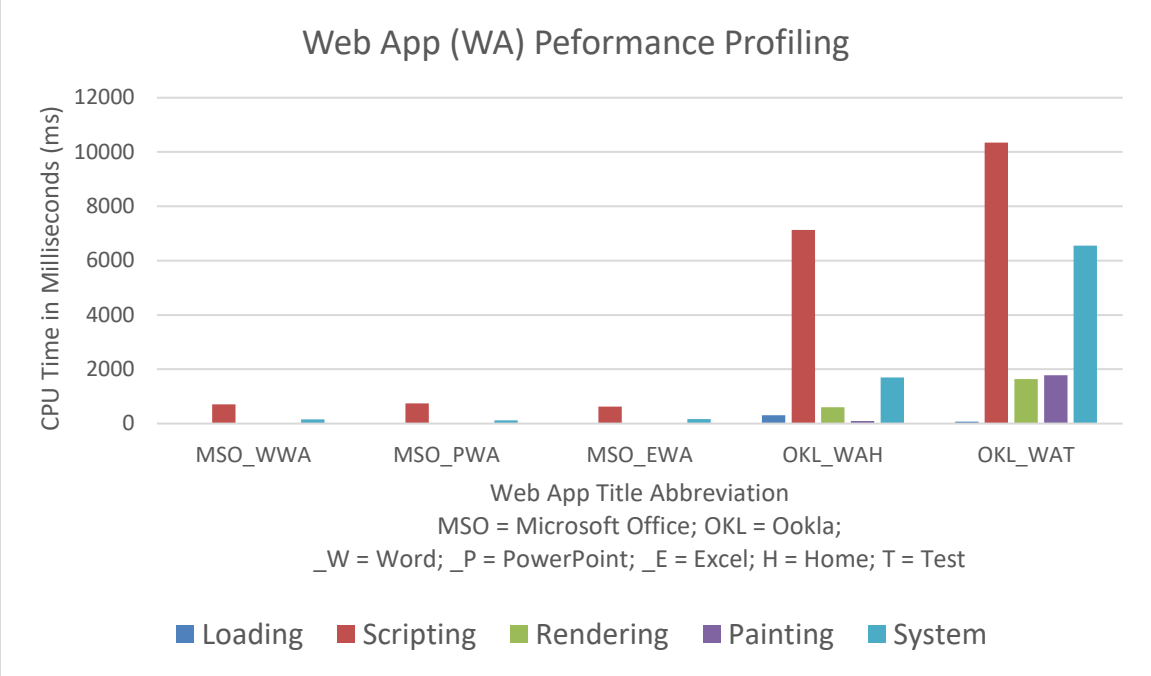

Figure 51: Web App Performance Profiling.

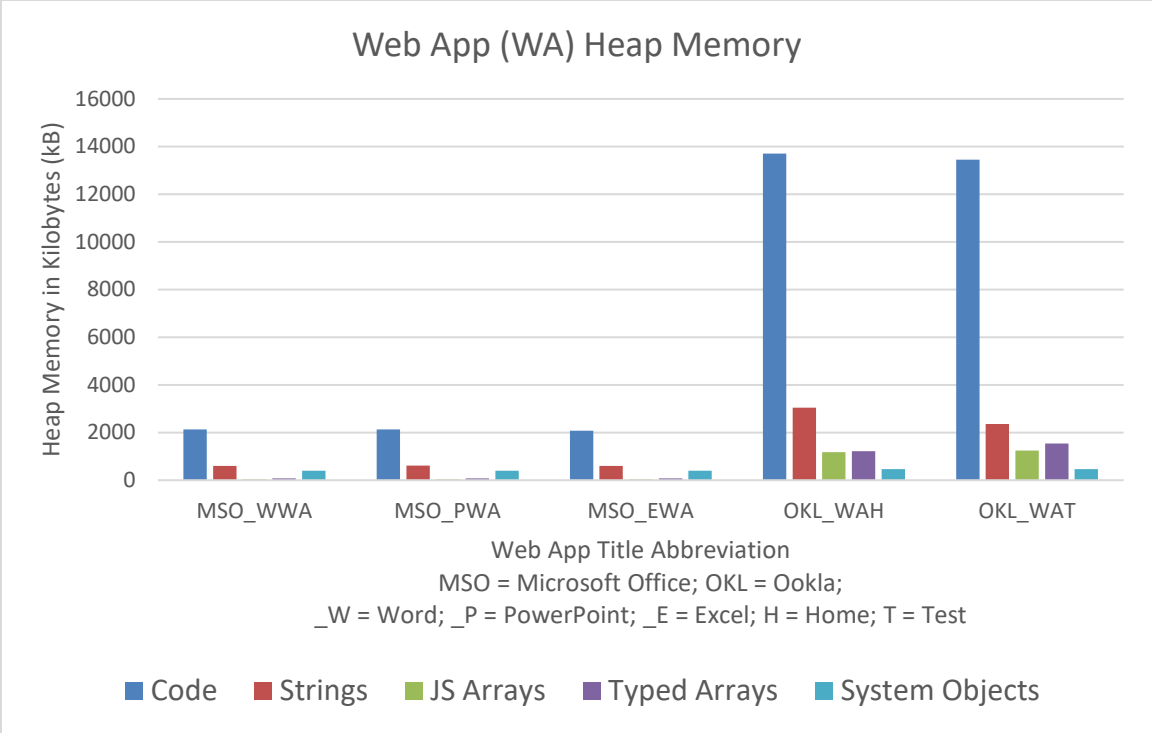

Figure 1: Web App Heap Memory.

Activity Monitor

CPU Usage. Out of the web apps tested Excel utilized the most threads at 21 with Word and PowerPoint both being a bit lower at 18 (Figure 53). Ookla had 15 when still and 14 when active, but that is within the margin of error.

Thread utilization increases substantially with the native versions with Word now taking the top spot of 75, and Power Point at 56 and Excel at 58 (Figure 54). Ookla sees a decrease from its still state at 7 but a slight increase in its active state at 17. This may point to Chrome processes not being able to scale resource usable as well as native apps for either increased efficiency or performance.

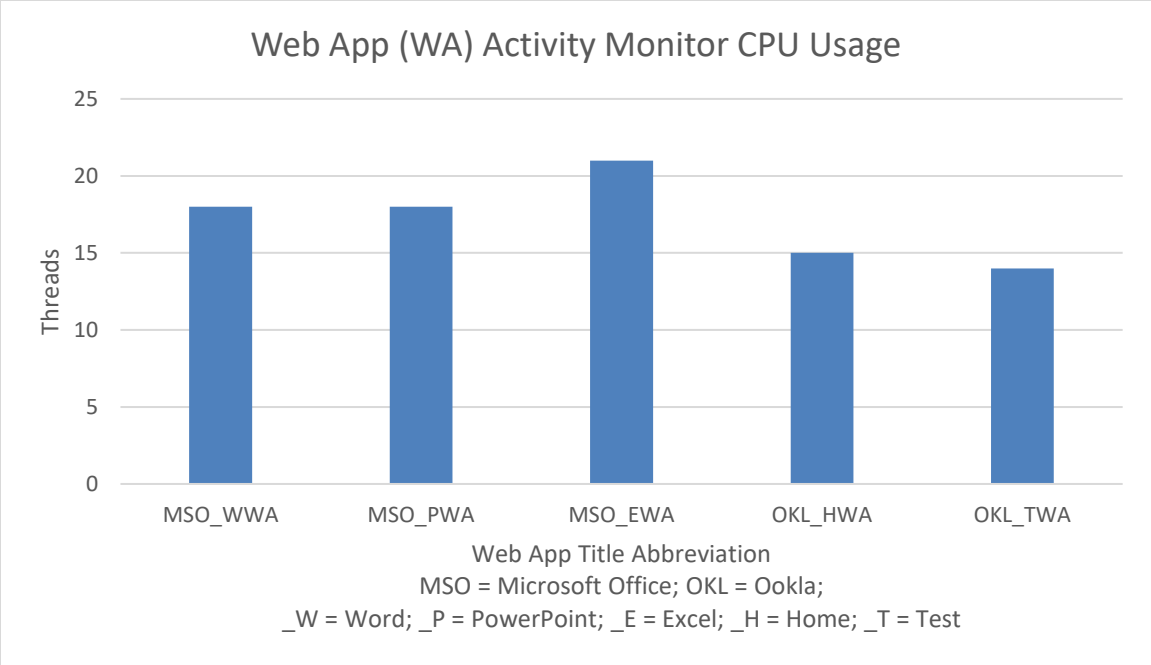

Figure 53: Web App Activity Monitor CPU Usage.

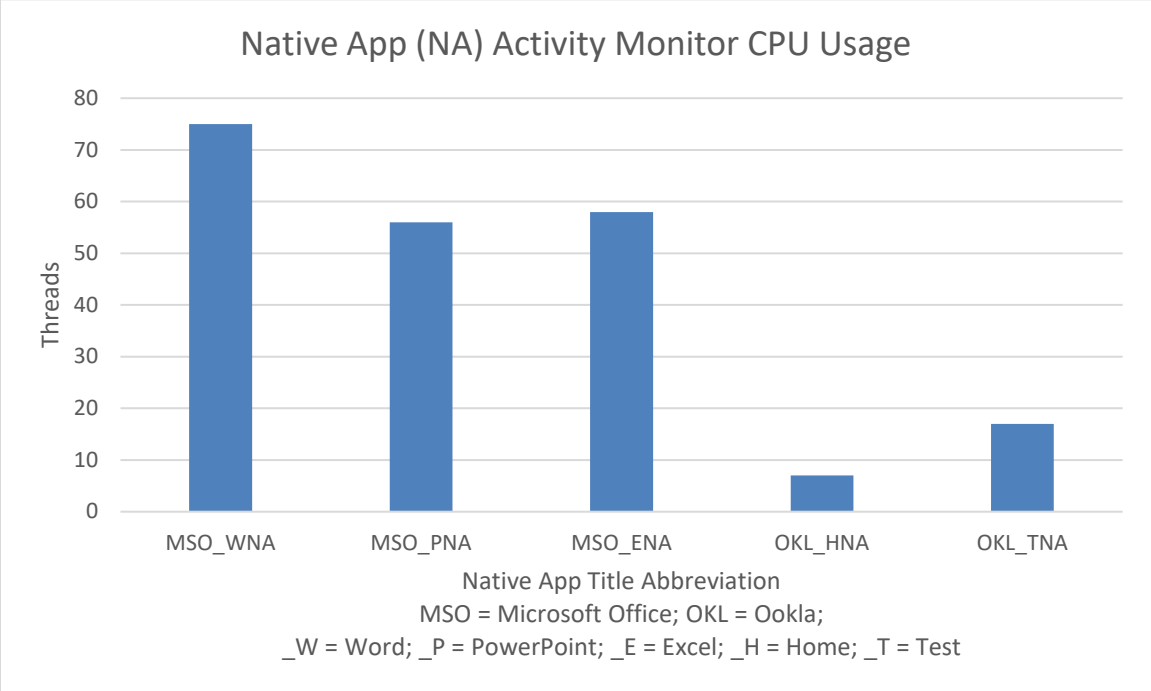

Figure 54: Native App Activity Monitor CPU Usage.

Sample Process. Collecting sample processes shows the difference between the current physical footprint and the peak physical footprint of memory for the Web Office Suite and Ookla still was not that large, with the exception of Word having around an 85

MB difference (Figure 55). The active Ookla process had the largest difference at around 130MB.

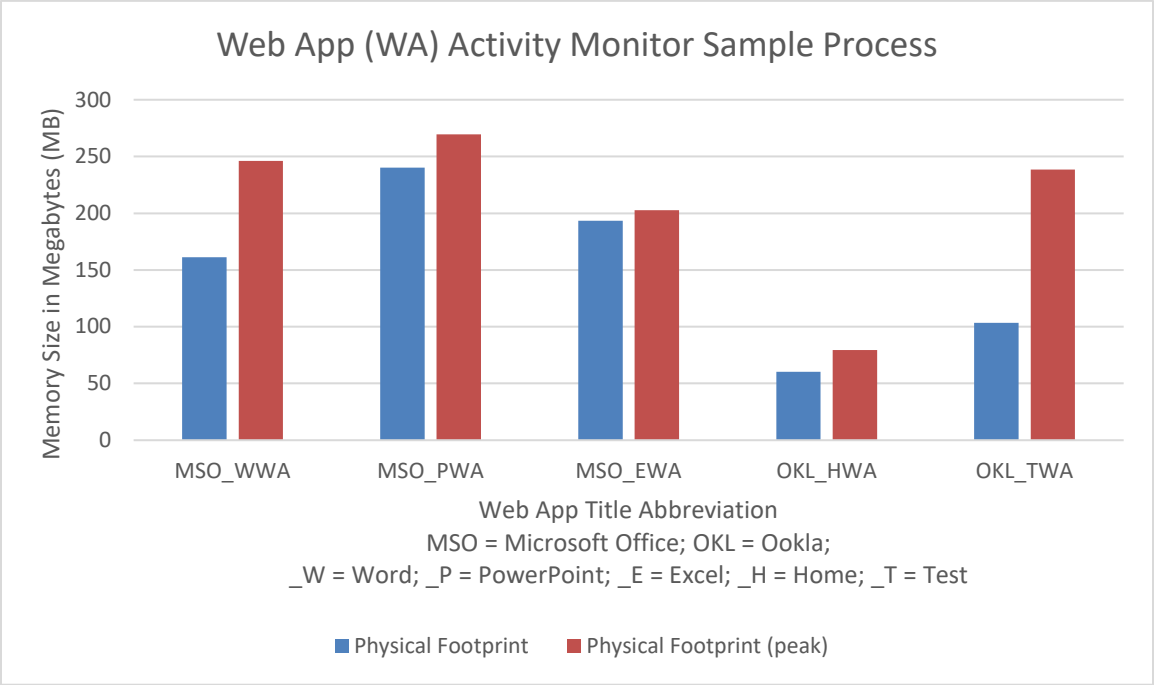

Figure 55: Web App Activity Monitor Sample Process.

Sample processes for the native apps show that their current and peak measurements are almost the same, even with the active Ookla process (Figure 56). The native Office Suite had a higher physical footprint all around compared to the web versions. Though this is not telling the whole story as there are other processes that ae running at the same time including the main Chrome process and other helper processes. If those are all taken into account, the web physical footprint might be higher overall. This idea can be carried over into the previous and future areas to be discussed.

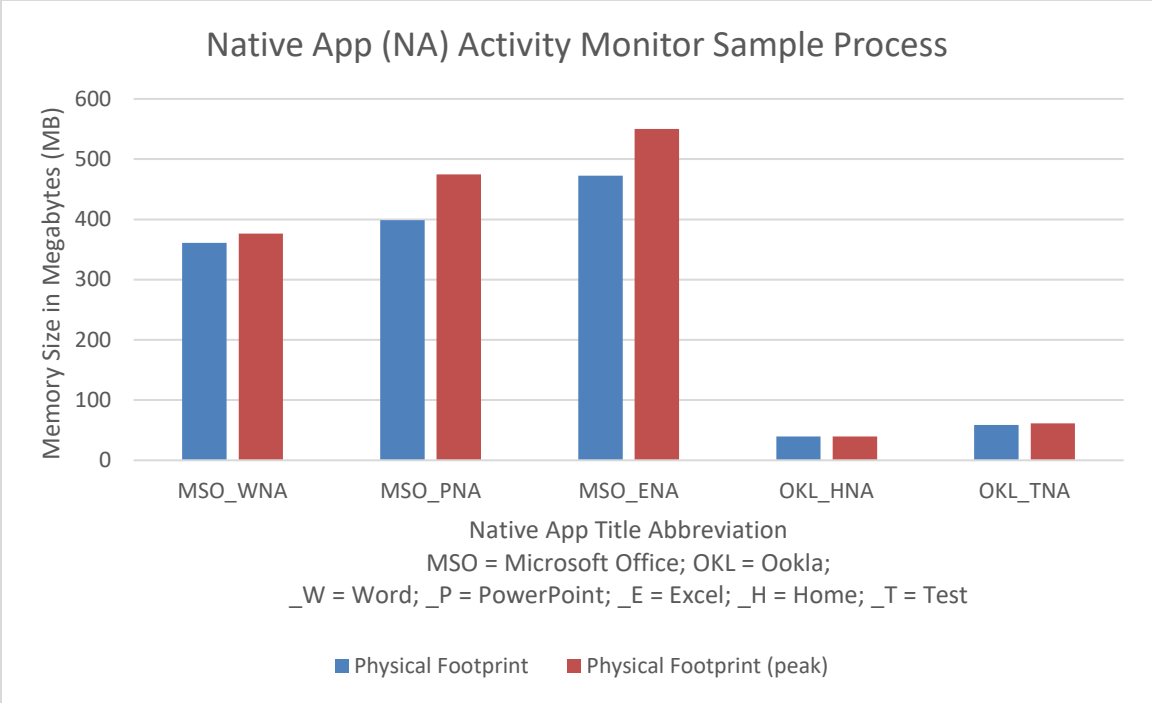

Figure 56: Native App Activity Monitor Sample Process.

Memory Usage. The active Ookla process, when compared to the still process, sees slight increases in memory and real-private memory, while a substantial increase is seen in real memory (Figure 57). The web office suite all follows the same pattern with Power Point using the most, then Word, and lastly Excel.

Ookla memory for the native apps is substantially lower than its web counterpart (Figure 58).

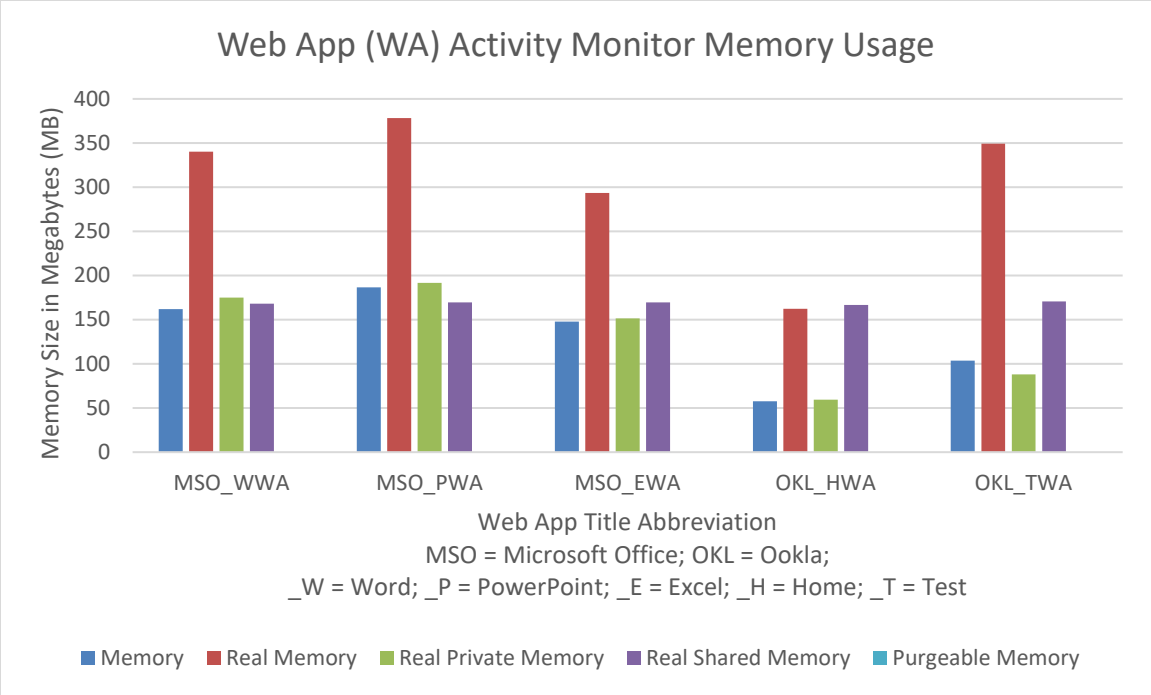

Figure 57: Web App Activity Monitor Memory Usage.

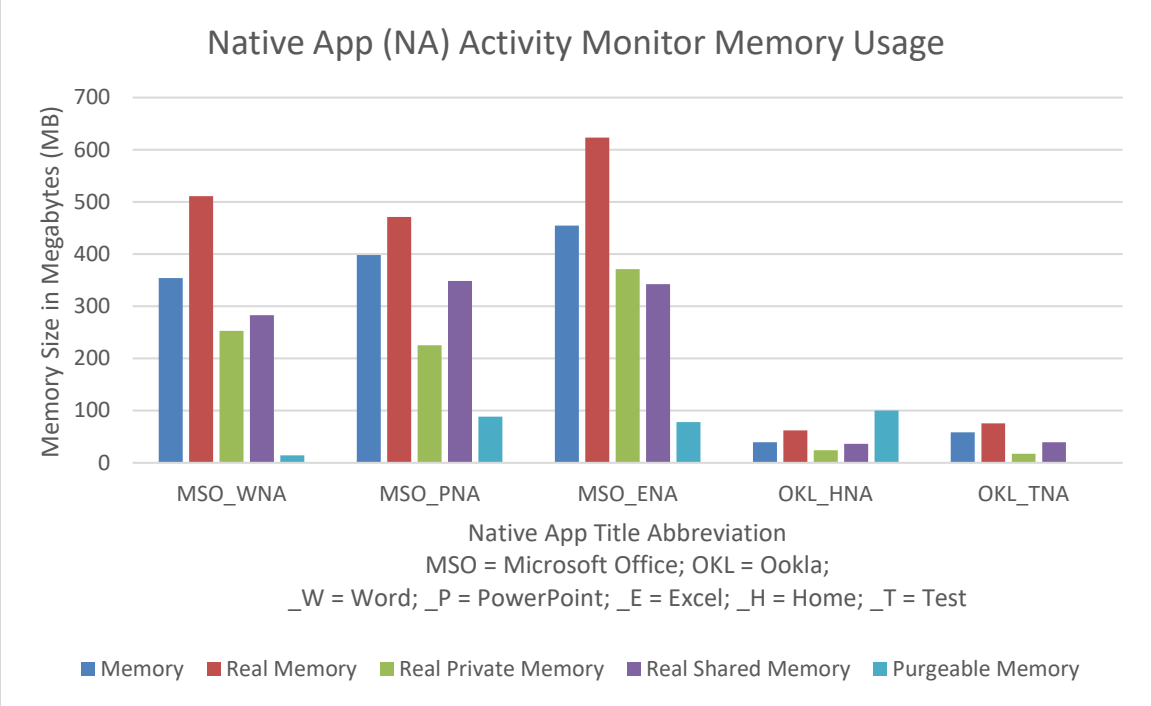

Figure 58: Native App Activity Monitor Memory Usage.

Energy Usage. Energy usage for the web processes barely charted, with all except Word having no data for the 12 hr Power (Figure 59). Unfortunately, the energy tabs

reading proved to be finicky, so there is a margin of error for the data and the 12 hr Power was not even prompted for the rest of the web apps.

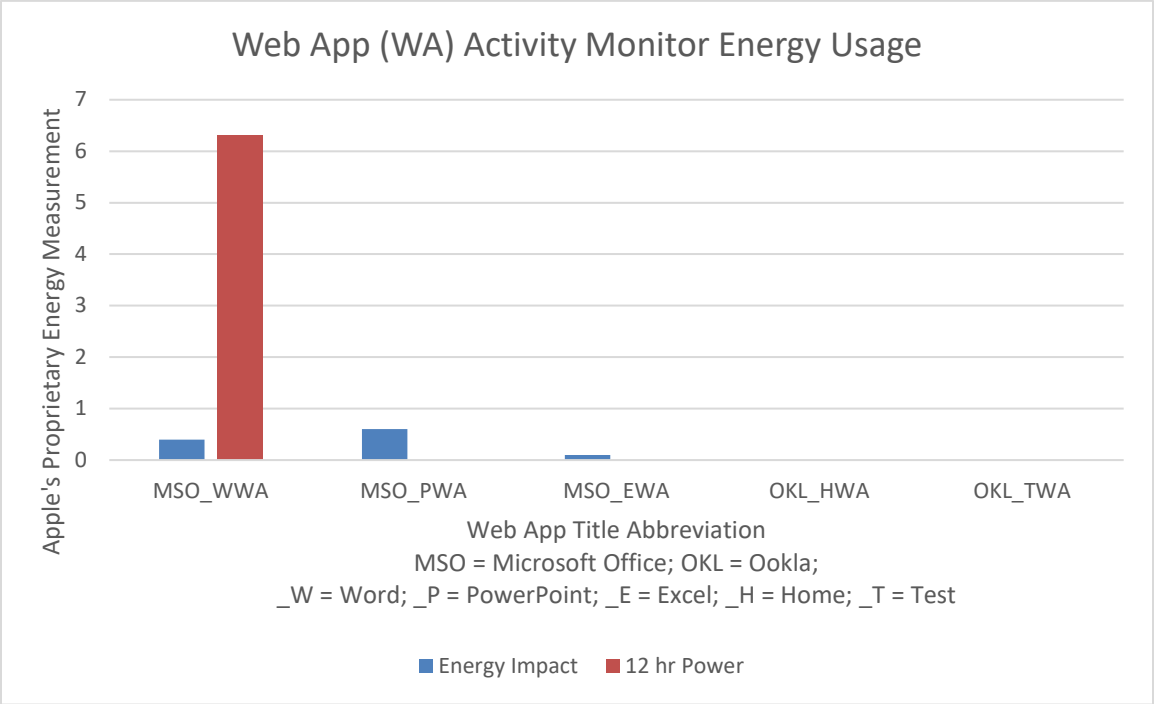

Figure 59: Web App Activity Monitor Energy Usage.

The native apps all chart data for 12 hr Power (Figure 60). Excel and most of all Power Point have the most Energy Impact eclipsing all the other processes.

Disk Usage. Web apps did not write any data to disk, it only read from it (Figure 61). Native apps for the most part read more data that they wrote, with the exception being Word (Figure 62). The still Ookla process is the only one of the native apps to have not written data to disk.

Network Usage. The web apps processes did not chart any data for network usage (Figure 63). It may be handled by the main process of a separate helper process. Out of the native app processes Word and Power Point by far sent and received the most. Word received more bytes than it sent, and Power Point sent more bytes than it received.

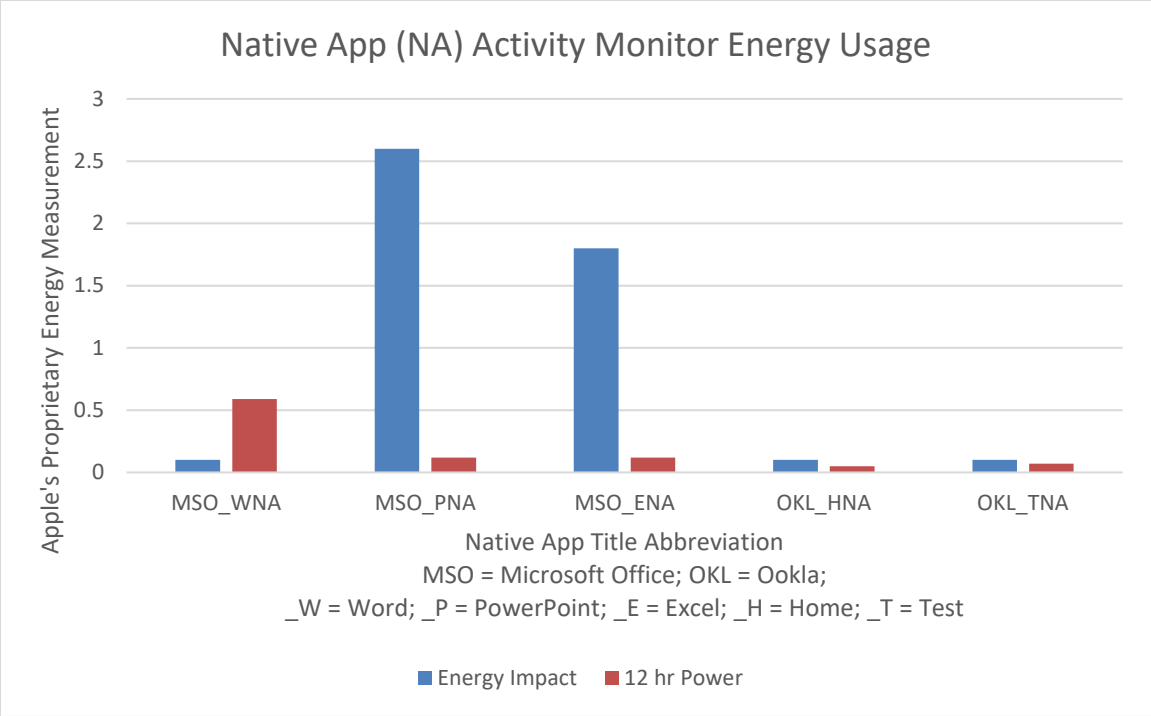

Figure 60: Native App Activity Monitor Energy Usage.

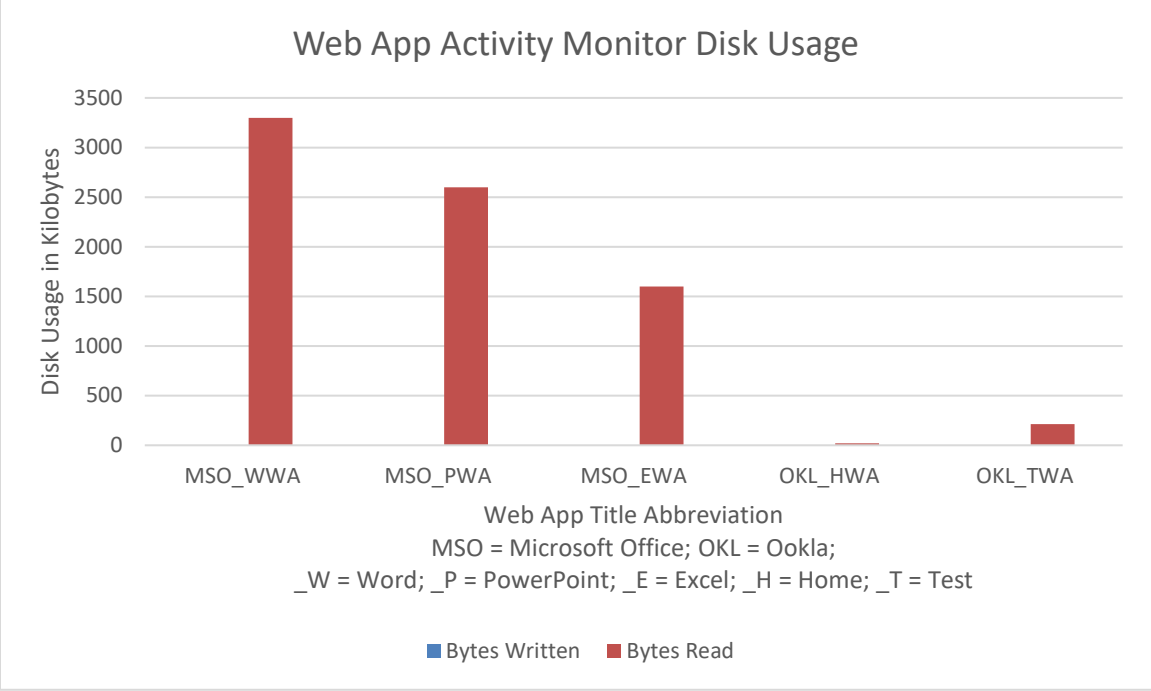

Figure 61: Web App Activity Monitor Disk Usage.

The network usage in packets follows the general trend of what is used the most but seems not to be affected by the number of bytes (Figure 64). For example, Word sent

received twice as many bytes than it sent but has a similar number of packets for each.

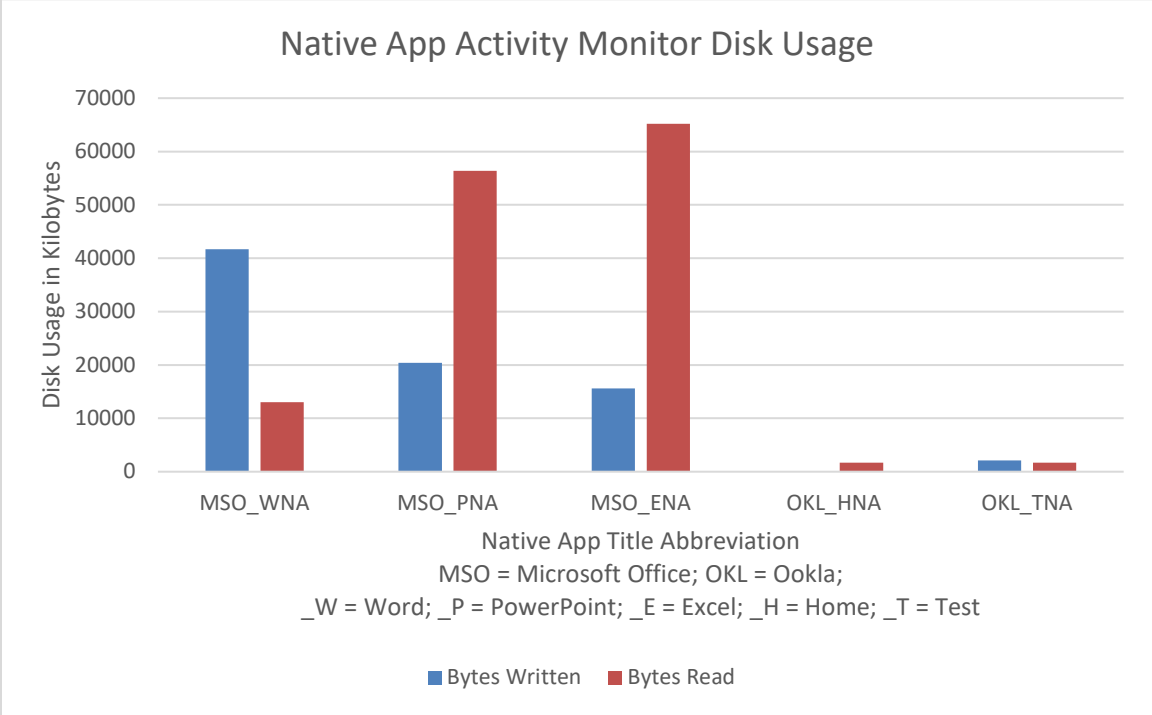

Figure 62:2 Native App Activity Monitor Disk Usage.

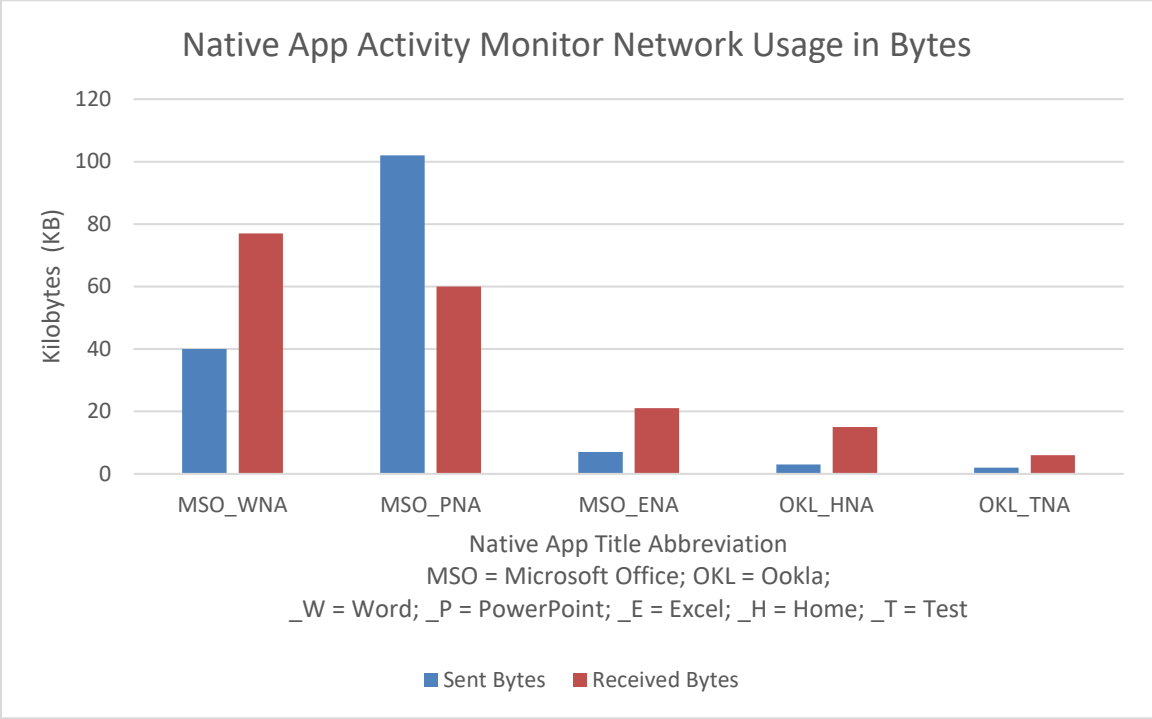

Figure 63: Native App Activity Monitor Network Usage in Bytes.

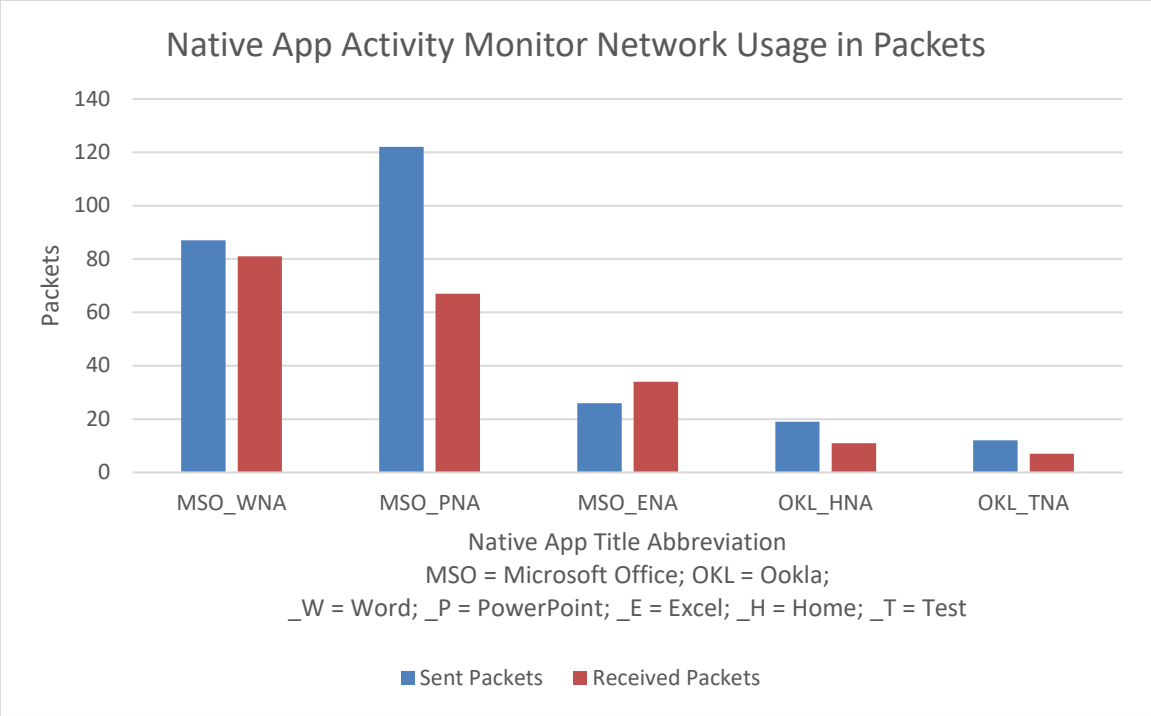

Figure 64: Native App Activity Monitor Network Usage in Packets.

Xcode CPU Profiler

Word. The web version of Word's processes only shows small spikes of activity compared to the native's large, sustained load for about the first 25 seconds when starting up and then switching to a sustained very low load on the CPU (Figure 65).

When using both actively the web process has a small, sustained load that still has noticeable segments where it stops, as opposed to the native version which had a slightly greater sustained load but had no noticeable segments (Figure 66).

Power Point. Power Point web still shows a small, sustained load with spikes to medium levels that are more frequent in the beginning and taper off as it goes on (Figure 67). The native has a medium sustained load for around 15 seconds then goes down to a small, sustained load with tiny periodical spikes.

Power Point web active shows a sustained medium load with spikes that get slightly more separated as it goes on (Figure 68). The native seems very similar but never

quite getting to medium levels of load with smaller peaks.

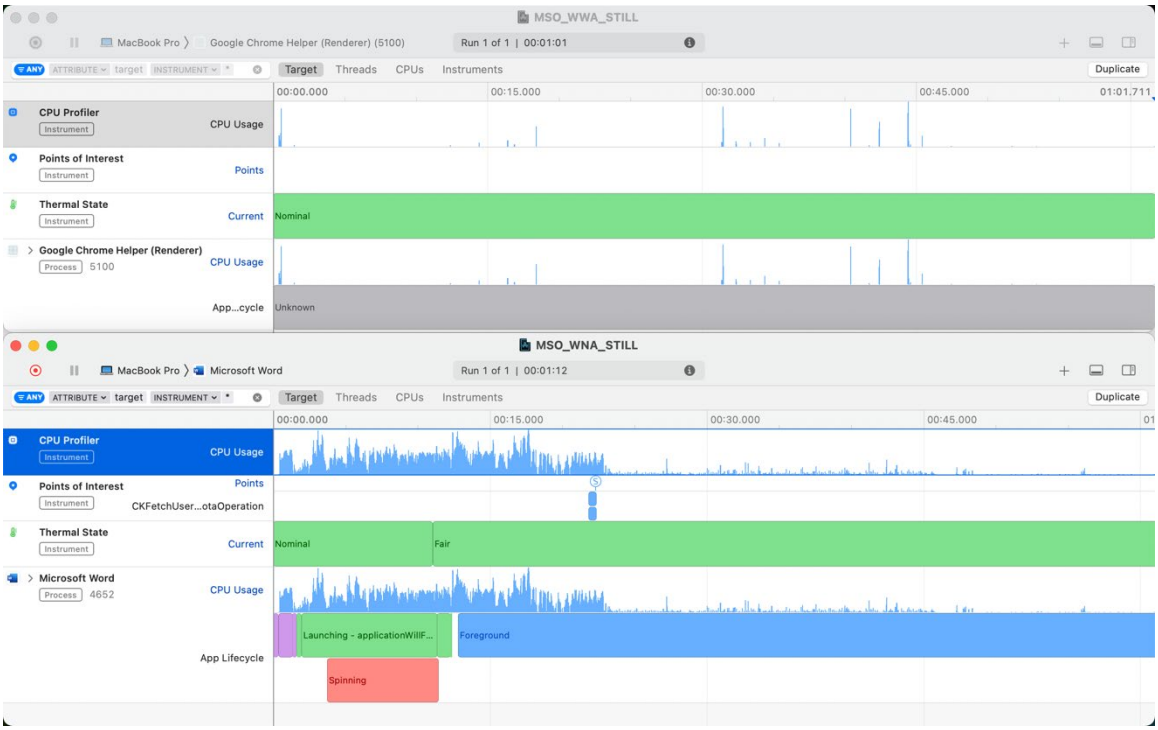

Figure 653: Word Still Web Vs. Native Xcode CPU Profiler.

Excel. Excel web still shows no sustained load with only large spikes at the beginning that are infrequent and turn into medium ones as it goes (Figure 69). The native has a medium sustained load until halfway through where it only has tiny spikes for the remainder.

The Excel web active seems to have segments of sustained loads starting with a large spike, then going to a medium load for the body of it, then going down to very low and repeat (Figure 70). The native has a medium load with large spikes in the beginning then going to a load just under medium with no spikes.

Ookla. The Ookla web version shows a medium sustained load with large spikes all the way throughout (Figure 71). The native has two very large spikes in the first couple of seconds then only has very tiny spikes for the remainder.

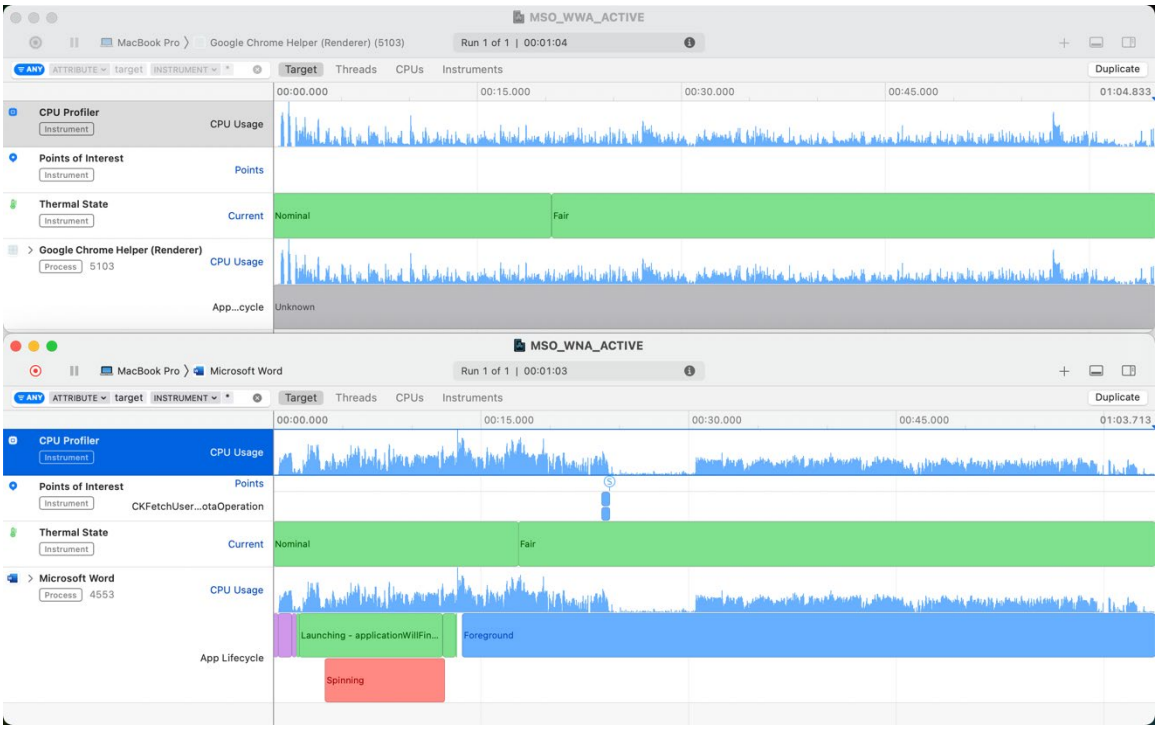

Figure 66: Word Active Web Vs. Native Xcode CPU Profiler.

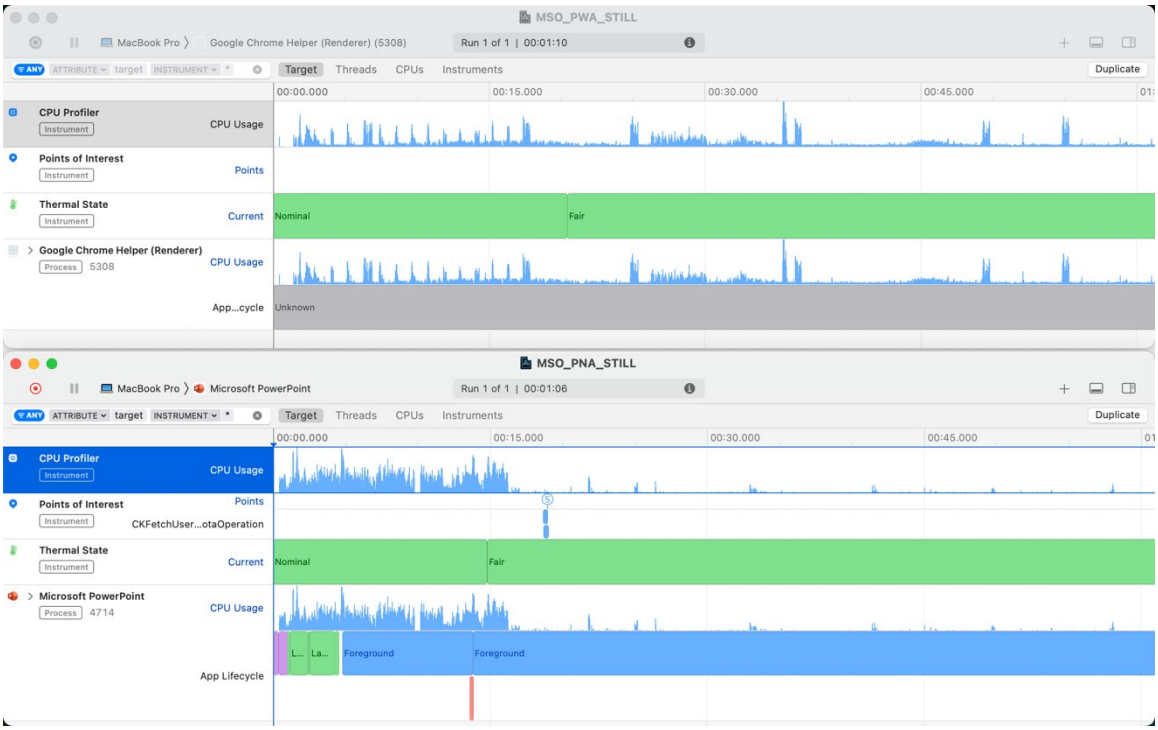

Figure 674: PowerPoint Still Web Vs. Native Xcode CPU Profiler.

Ookla web active shows a small, sustained load with large spikes in the very beginning and late middle, then dropping the sustained load with just small spikes (Figure

72). The native has two large spikes in the very beginning, the small, sustained load with medium as well as large spikes, and then only a handful of small spikes.

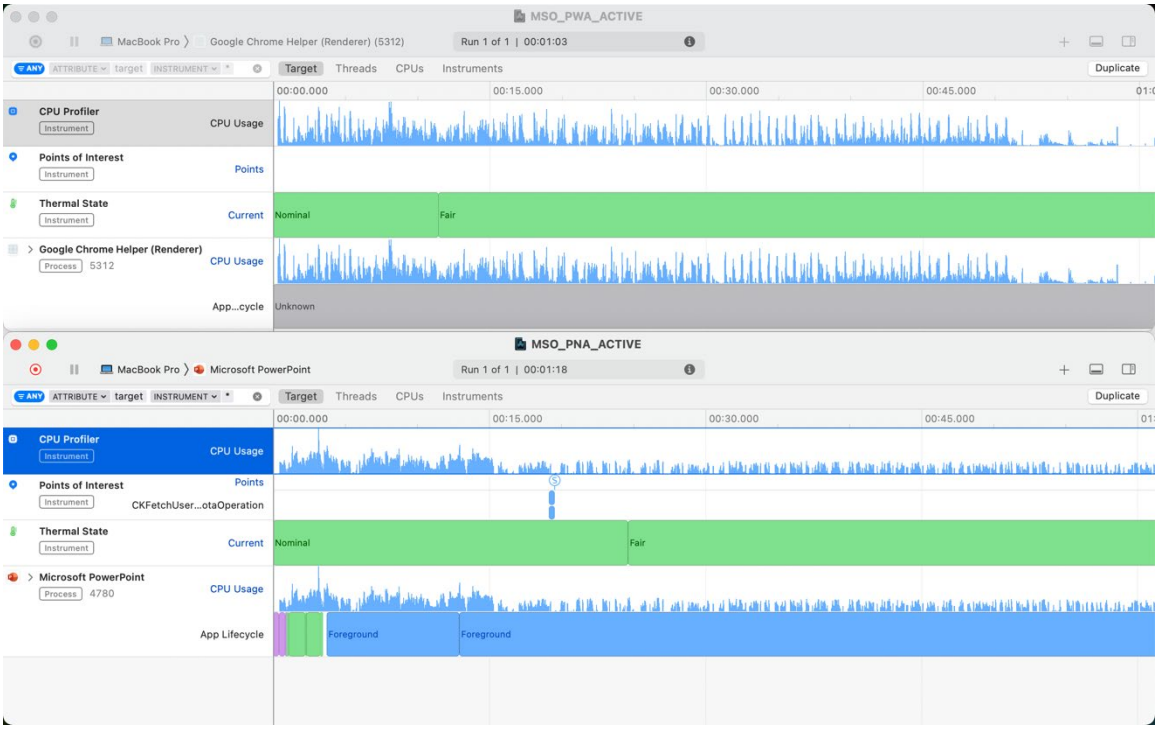

Figure 685: PowerPoint Active Web Vs. Native Xcode CPU Profiler.

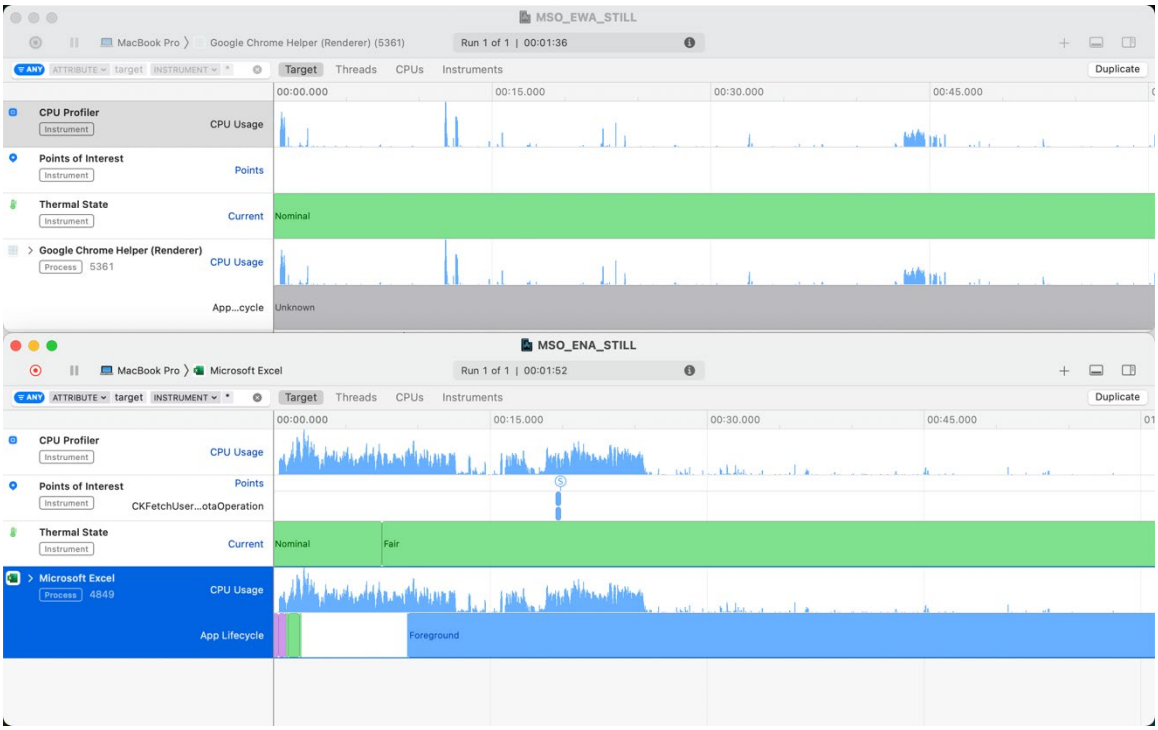

Figure 69: Excel Still Web Vs. Native Xcode CPU Profiler.

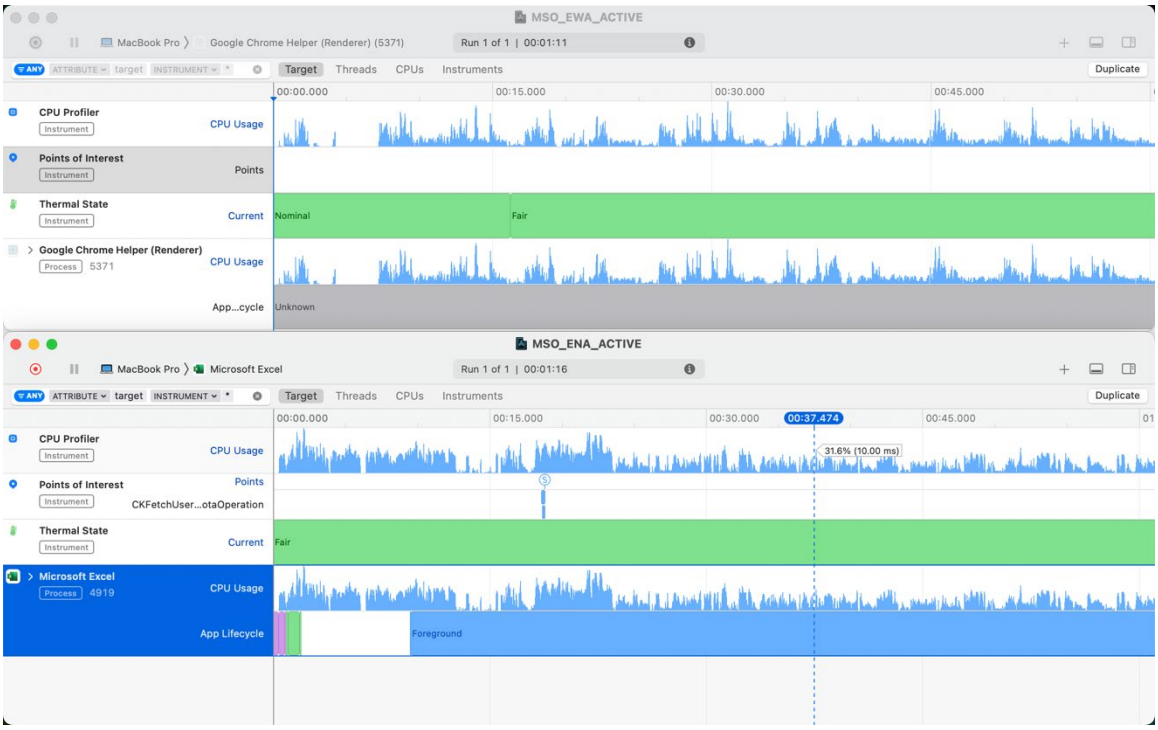

Figure 706: Excel Active Web Vs. Native Xcode CPU Profiler.

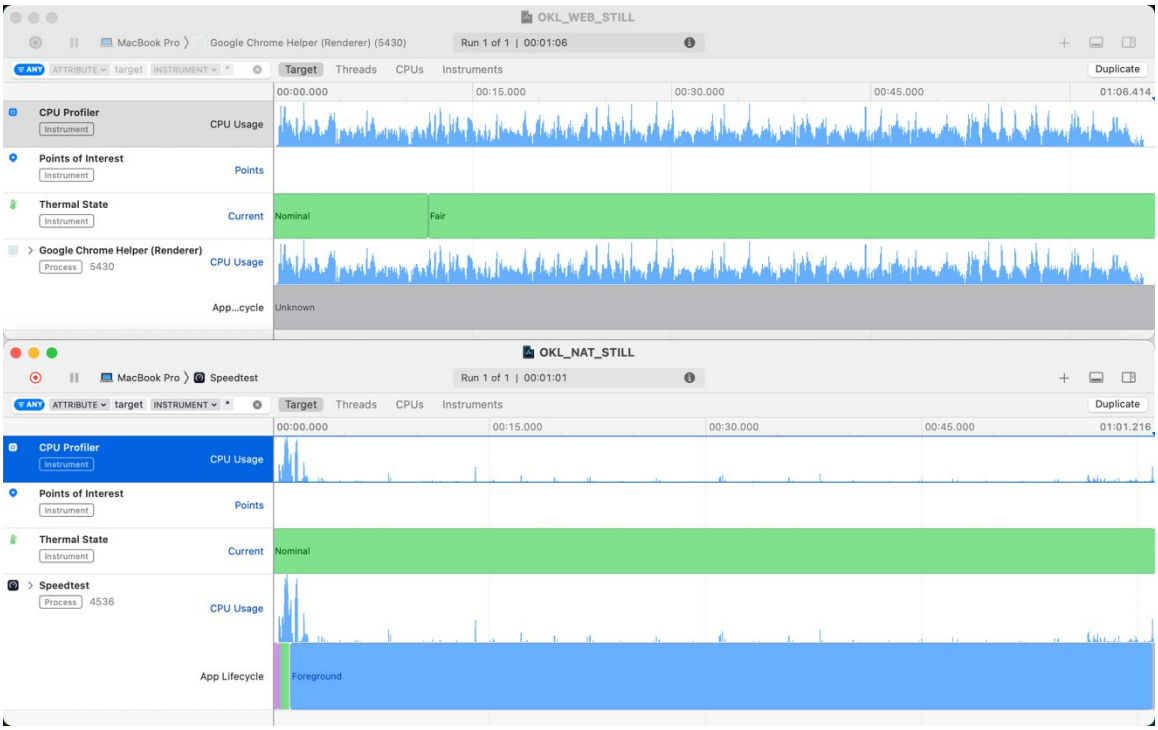

Figure 7: Ookla Still Web Vs. Native Xcode CPU Profiler.

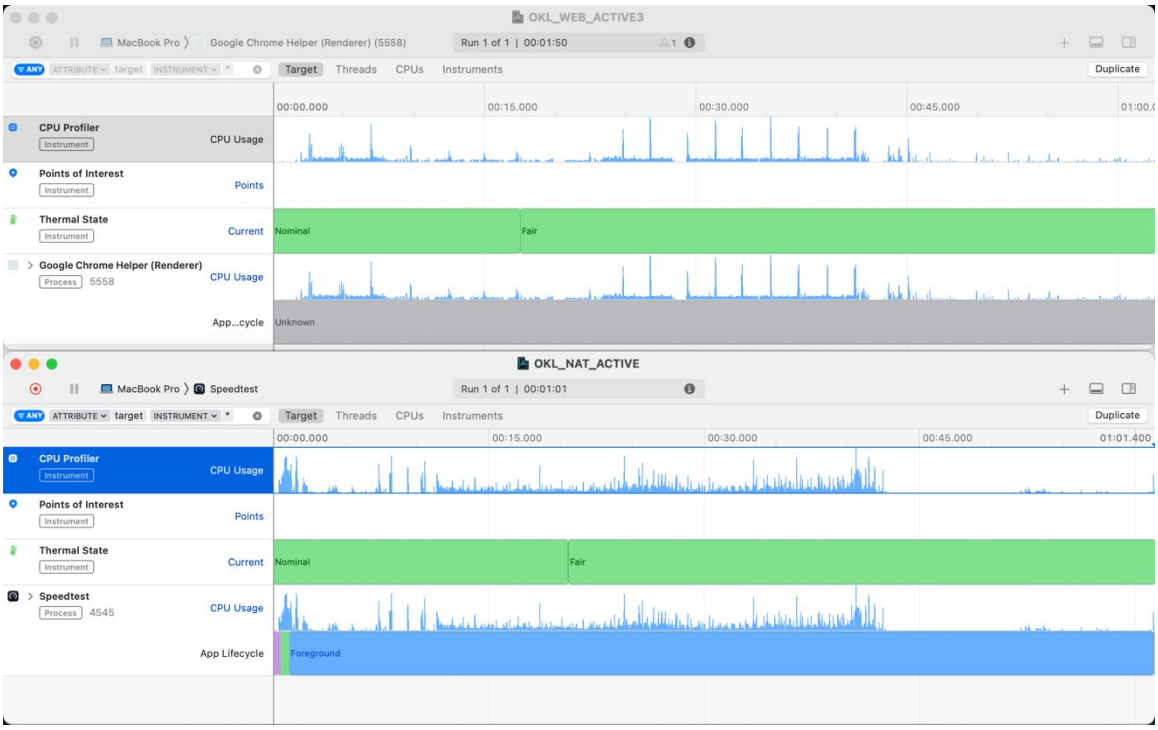

Figure 72: Ookla Active Web Vs. Native Xcode CPU Profiler.

Discussion

Limitations. One limitation of this research is the profiling of web apps in Google Chrome with the activity monitor cannot give a complete picture. Only one process can be targeted at a time and the browser has a plethora of helper processes active even with one tab open. The process that was doing the most work was able to be identified for the data collection but the real-world combined resource usage of the web app, its helper processes, and the main browser process, which is not cumulative of the others, is likely worse.

Another limitation is the use of the energy section of the activity monitor. Not only does the section not register any readings for one of the data points, being 12-hour power, but it is in an Apple proprietary format. Thankfully for the purposes of this research is just used as a comparative measurement between web apps and native apps. This still allowed the collection of useful information but in the future a different

software tool or a hardware tool for reading power be used if the data needs to be extrapolated further.

Implications. After completing a second data collection, analyzing the results, and comparing them to the first collection there is both good and bad. Some of the mainstream sites have actually shown some improvement while the alternative sites have gotten ever so slightly worse in their resource usage.

When comparing the main processes of the web and native apps, while on the surface it seems like the native uses more resources making so it is the worse version, the web version relies on the main chrome process and other helper processes that take up just as many resources possibly if not more than the native. Also, it does not put into perspective how the apps perform in terms of general usage considering the number of resources it uses.

The aspects taking up the most resources are the scripting and system for the CPU then code and typed arrays for the memory. There are some outliers in the data but most of the websites follow that general pattern. These biggest resources could be shrunk, possibly besides typed arrays which are more efficient than the alternative of java arrays.

Software is less optimized because of the preference by developers for ease of development and deployment. Websites and web apps will not be able to really use the hardware to its fullest capabilities due to the layer of abstraction the web browser adds. And as seen in the data purgeable memory was not found to be in any of the web apps, so the OS itself cannot easily clear memory no longer needed by the browser. This combined with the many extra helper processes for each tab that's open means that users are not able to open as many tabs and different websites as they need.

Based on the results shown there are some recommendations that can be given to

developers and users. For the former Scripting could be reduced by using less JavaScript or using another more efficient web language. System could be reduced by not applying tracking code to the website and using only what is necessary for functionality of the website. Code could be shrunk by doing periodical rewrites that help to eliminate any spaghetti code keeping it small. Developers should also investigate having their own native apps on OS's, especially ones that already have native mobile apps on iPhone and Android. Users should look at alternatives based on their usage patterns. For example, if they have a lot of search engine tabs open, opt for one that has nearly as good of results as DuckDuckGo which uses a fraction of the resources compared to Google.

Chapter 6: Conclusion

Software has increased its capabilities and performance over time, along with hardware to open new possibilities. But it seems that there has been a stagnation in what software is capable of and the tradeoff of ease of development over best possible performance. The shift from desktop to mobile, the use of high-level tools in development, and the resulting increase of resource use has shown this. Using modern browsers and OS analytic tool data was able to be extracted to give insight into what more specifically is happening and is there a simple solution, creating a middle ground for developers and users. Looking at some the most popular categories for websites, as well as applications with web and native versions, different aspects were able to be identified for the most resource use. Some aspects that were the highest have little possibility of being lessened, but a good amount with the right philosophy in mind easily could be. For the end users for most sites and applications there are alternatives that will help to make their tasks smoother, ultimately improve the quality of the end experience.

References

Apple, Kerris, N., & Dowling, S. (2007, January 9). *Apple Reinvents the Phone with iPhone*. Apple Newsroom. https://www.apple.com/newsroom/2007/01/09Apple-Reinvents-the-Phone-with-iPhone/

Aqeel, W., Chandrasekaran, B., Feldmann, A., & Maggs, B. M. (2020). On Landing and Internal Web Pages: The Strange Case of Jekyll and Hyde in Web Performance Measurement. *Proceedings of the ACM Internet Measurement Conference*, 680–695. https://doi.org/10.1145/3419394.3423626

Blackburn, S. M., Cheng, P., & McKinley, K. S. (2004). Myths and realities: The performance impact of garbage collection. *ACM SIGMETRICS Performance Evaluation Review*, 32(1), 25–36. https://doi.org/10.1145/1012888.1005693

Britannica. (2024). *Computer—ENIAC, Electronic, Computing | Britannica*. https://www.britannica.com/technology/computer/ENIAC

Cen, L., Marcus, R., Mao, H., Gottschlich, J., Alizadeh, M., & Kraska, T. (2020). Learned garbage collection. *Proceedings of the 4th ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, 38–44. https://doi.org/10.1145/3394450.3397469

Cho, S., Park, C., Won, Y., Kang, S., Cha, J., Yoon, S., & Choi, J. (2015). Design Tradeoffs of SSDs: From Energy Consumption's Perspective. *ACM Transactions on Storage*, 11(2), 1–24. https://doi.org/10.1145/2644818

Computer History Museum. (2024). Apple II - CHM Revolution. https://www.computerhistory.org/revolution/personal-computers/17/300/1047#:~:text=4%201%2F2%20x%2015%201%2F2%20x%2018%20in.&text=Apple%20Computer%2C%20Inc.&text=Steve%20Wozniak%20d

esigned%20the%20Apple,BASIC%20language%20in%20permanent%20memory.

Dembrow, B. (2022). Investing in Human Futures: How Big Tech and Social Media Giants Abuse Privacy and Manipulate Consumerism. *University of Miami Business Law Review*, 30(3), 324.

Dornauer, B., & Felderer, M. (2023). *Energy-Saving Strategies for Mobile Web Apps and their Measurement: Results from a Decade of Research* (arXiv:2304.01646). arXiv. http://arxiv.org/abs/2304.01646

Eloon C. Hall. (1972). *MITs Role in the Apollo Project: Computer Subsystem*. http://ibiblio.org/apollo/hrst/archive/1029.pdf

Gunnarsson, K., & Herber, O. (2020). *The Most Popular Programming Languages of GitHub's Trending Repositories*. https://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-280113

Hoakley. (2022, August 13). Activity Monitor: Meanings and misleadings. *The Eclectic Light Company*. https://eclecticlight.co/2022/08/13/activity-monitor-meanings-and-misleadings/

IEEE. (2023, April 18). *ENIAC: The World's First Computer*. IEEE Life Members. https://life.ieee.org/eniac-the-worlds-first-computer/

Intel. (2024). *What Is Intel® Turbo Boost Technology?* Intel. https://www.intel.com/content/www/us/en/gaming/resources/turbo-boost.html

Ismail, M., & Suh, G. E. (2018). Quantitative Overhead Analysis for Python. *2018 IEEE International Symposium on Workload Characterization (IISWC)*, 36–47. https://doi.org/10.1109/IISWC.2018.8573512

Jennings, P., & Brewster, T. (1998). *The Century* (1st ed). Doubleday.

Johnson, T., & Seeling, P. (2013). *Desktop and Mobile Web Page Comparison: Characteristics, Trends, and Implications* (arXiv:1309.1792). arXiv. http://arxiv.org/abs/1309.1792

Ladan, Z. (2015). *Comparing performance between plain JavaScript and popular JavaScript frameworks*.

Mighell, T. (2019). The State of the Web Browser, 2019. *Technology: The Digital Toolkit*, 32–33.

National Museum of American History. (2024). *Apple II Microcomputer*. https://americanhistory.si.edu/collections/nmah_334638

Pang, C., Hindle, A., Adams, B., & Hassan, A. E. (2016). What Do Programmers Know about Software Energy Consumption? *IEEE Software*, 33(3), 83–89. https://doi.org/10.1109/MS.2015.83

Pinto, G., & Castor, F. (2017). Energy efficiency: A new concern for application software developers. *Communications of the ACM*, 60(12), 68–75. https://doi.org/10.1145/3154384

Saleem, J. J., & Weiler, D. T. (2018). Performance, workload, and usability in a multiscreen, multi-device, information-rich environment. *PeerJ Computer Science*, 4, e162. https://doi.org/10.7717/peerj-cs.162

Selakovic, M., & Pradel, M. (2016). Performance issues and optimizations in JavaScript: An empirical study. *Proceedings of the 38th International Conference on Software Engineering*, 61–72. https://doi.org/10.1145/2884781.2884829

Shmueli, O., & Ronen, B. (2017). Excessive software development: Practices and penalties. *International Journal of Project Management*, 35(1), 13–27. https://doi.org/10.1016/j.ijproman.2016.10.002

Statista. (2024, March). *Recorded music market revenue worldwide 2023*. Statista. https://www.statista.com/statistics/292081/music-revenue-worldwide-by-source/

University of Michigan. (2024, March 14). *ENIAC Display | Computer Science & Engineering at Michigan*. Computer Science and Engineering. https://cse.engin.umich.edu/about/beyster-building/eniac-display//

Vailshery, L. S. (2024, March 4). *Internet browser market share 2012-2024*. Statista. https://www.statista.com/statistics/268254/market-share-of-internet-browsers-worldwide-since-2009/

Worldwide Business Research. (2019). *The Amazon Effect & What it Means for Retailers.* Future Digital Finance 2025. https://futuredigitalfinance.wbresearch.com/blog/rebalancing-retail-amazon-research-strategy