Evaluating Caching Strategies in PWA for Performance and Energy Efficiency

Gamma Andhika
2669061
VU Amsterdam
g.a.o.octafarras@student.vu.nl

Neeraj Sathyan
2660192
VU Amsterdam
n.sathyan@student.vu.nl

Tavneet Singh
2668222
VU Amsterdam
t.s.tavneet@student.vu.nl

ABSTRACT

Progressive Web Applications (PWAs) built using common web technologies like HTML, JavaScript and CSS are gaining popularity as an alternative to native apps and web based apps. Caching in PWAs is used by developers to provide a personal experience to the user while the device is offline and there exist multiples types of caching strategies for PWAs.
Goal. The paper aims to compare different caching strategies of PWAs across energy consumption, performance and network load.
Method. We conducted an empirical study targeting 9 third party applications. The experiment is 1 factor Greater Than 2 Treatments with Caching strategy being the single factor and type of caching strategy used as the treatments(stale-while-revalidate, cache first then network, network only). The response variables of the experiment are i)Energy Consumption ii)CPU Usage iii)Memory Usage iv)Network Bandwidth Size and v)Count of Network Requests. The experiments are conducted on an Android device running Google Chrome[1] browser.
Results. The results of our experiment suggest that the energy consumption, PWA performance and PWA network load all vary according to the type of caching strategy being implemented for the PWA with different strategies having varying magnitudes of effect on different parameters.
Conclusions. This study confirms the base hypotheses that different caching strategies have varied influence over the metrics of different types of PWAs which can be useful for PWA developers to decide upon the type of strategy for their application which would lead to the best optimization and enhance the overall efficiency of their applications.

1 INTRODUCTION

With the pervasiveness of mobile phones, smartphone application development is booming. It is estimated that there are now about 3.5 billion users with access to the internet using a smartphone [1]. There is an explosion of mobile applications as mobile devices have become more powerful over the years. The devices now support applications dedicated to gaming, virtual reality, photo, and video editing, dynamic web browsing, and much more. But these dedicated applications are built-in native SDKs like Android Studio to utilize the functionalities the smartphone can provide like notifications and sensor readings and app caching for offline functionality.

Apart from dedicated applications, the field of web apps has not been as popular due to a lack of offline availability and user personalization. This changed with the introduction of Progressive Web Apps (PWAs). PWAs first introduced in a blog post by Alex Russel[2] are a set of strategies, APIs, and techniques to provide the native app experience in an offline way. A PWA according to Hume et. al. [3] is:

- Responsive
- Connectivity-independent
- Interactive with a feel like a native app's
- Always up-to-date
- Safe
- Discoverable
- Re-engageable
- Installable
- Linkable

PWAs can be installed and distributed from browsers without reliance on app stores and require very less development time compared to their native counter-parts[4]. A core essential component that helps in linking the web browser and functionalities together to form the PWAs are the service workers. Service workers are a set of JavaScript APIs that run separately from the main thread acting as a buffer between the network and the document renderer, which allows them to render documents even if there is no access to the network.[5] They can also serve as a cache and network proxy [6]. PWAs are also shown to have a higher user retention rate due to their simplicity in installing, for example, if a website such as Twitter offers a PWA feature they can be installed simply by accessing the browser menu and selecting the installation option[7].

This paper will discuss in-depth the caching strategies in PWAs and their impact on performance and energy consumption in the mobile device. The need for cache-based web applications is crucial for enhancing and stabilizing user experience. Although mobile devices are becoming more powerful, the mobile network always finds it difficult to keep up with the ever-growing demand of huge user bases and this affects the bandwidth [3]. From 2G to 3G and now 4G, the network is highly unreliable and shows high latency in high dense areas as there are more users connected to limited bandwidth. Web applications and even browsing become a gruesome task for the users to explore in these restricted environments. Thus caches are one of the core functionalities provided by PWAs to make their usage closer to the native applications.

This paper focuses on an empirical study to evaluate the impact of PWA caching on performance and energy consumption of PWAs based on different caching strategies in android devices. The **goal** of this experiment is to quantify the effect of different caching strategies for PWAs on device energy consumption and performance.[8] The experiment is motivated by the results of [9]. The goals are also motivated based on the fact that mobile devices are becoming the de-facto ubiquitous computing device for a human and thus there are massive opportunities to use the device to perform the

[1]https://www.google.com/chrome/

computational task to the level expected by a desktop computer but with less energy [10]. Malavolta et. al. [9] provided a result that there is no further energy drain by using the caching functionality in a PWA. This indicates the high potential of usage of caches in PWAs to increase user experience without compromising on the amount of energy needed to do the extra functionality. This in turn opens up opportunities for more sustainable development of software. Malavolta et. al. focussed on results which any web app developer could utilize to develop or utilize the appropriate caching strategy based on performance and energy efficiency in PWAs.

1.1 Background

Caching Strategies. A Cache object represents a request response list. The caching strategies apply when requests come into the service worker from the user [8] post-installation usage of the app. Some examples of caching strategies in PWAs based on [8] are:

- Stale-While-Revalidate: Upon a request to the service worker, check for the response in the cache. For a cache hit, the response is delivered to the app, and the cache is revalidated. For a cache miss, the response is retrieved from the network and delivered to the cache. Critical files with higher priority are retrieved using this method in PWAs like APIs to get current weather data.
- Cache First Then Network: The service worker first gets the response from the cache. For a cache miss, it fetches the response from the network and then serves it to the app before storing it into the cache. On a cache hit the response is served into the app, without any revalidation with the network. This strategy is mostly used for loading assets for a web component like images, CSS, and other non-critical files.
- Network First Then Cache: Exact opposite of *Cache First Then Network*. A JavaScript code snippet of the strategy is shown in listing 1. Requests with frequent updates like POST requests are utilized under this strategy in PWAs. *Cache First Then Network*
- Cache Only: The response is taken from the cache alone. This works completely offline without any interference with the network. Assets that never change in an app's lifetime like an app's logo, developer names are cached the first time and never updated via the network.
- Network Only: Service worker solely interacts with the network to get responses for the requests. Assets that change over real-time use of this strategy.
- Multi-Strategy: A PWA can take advantage of all these strategies and hence can be found that some PWAs use more than one strategy for different request types like Tinder using Cache First for their images and Network First for text data. Multi-Strategy Apps will not be used for this experiment.

```
//add event listener for fetch event
self.addEventListener('fetch', event => {
//respond to event
  event.respondWith(
  //fetch the request over the network
     fetch(event.request).then(response => {
       //putting the response in the cache
        cache.put(event.request, response.clone())
            ;
        //returning the response
        return response;
     }).catch(_ => {
       return caches.match(event.request);
     });
   );
});
```

Listing 1: Code for Network First then Cache Strategy

2 RELATED WORK

PWAs are a relatively new technology with the initial research focussed on using them as an alternative to native mobile apps and web apps [11, 12]. Fortunato et. al talk about the low size footprint of PWAs as compared to the native apps while providing all the basic functionalities which made them popular in areas with low internet connectivity. Besides this, there is a lack of research in PWAs especially when it comes to measuring performance and energy efficiency.

The work done Malavolta et.al [9] is the most relevant to our study where the authors worked on assessing the energy efficiency and performance of 9 PWAs on an Android device. They compared the energy utilisation and page load times of the PWAs with an empty and populated cache. Our work can be considered as an extension of the study as we also focus on energy efficiency and performance but i) Our study will compare the metrics across different caching strategies instead of empty vs populated cache methodology used in their approach. ii) Our performance metrics are different (CPU, memory and used bandwidth vs page load times)

Battery is a limited resource on mobiles and caching has been proposed to improve the energy efficiency of the devices[13]. Work on energy efficiency of PWAs was done by Malavolta et. al [14] where they studied the impact of service workers on Energy Efficiency of PWAs across multiple devices and different network conditions. Their results showed that service workers did not have a major impact on energy consumption. Our work is different as i) we also focus on performance metrics and i) we focus on the initial load while their energy efficiency measurement was spread out across time on various scenarios.

In prior work done on desktop clients, it was observed that caching improves latency by reducing page load times in web pages[15, 16]. Vesnua et al. and Wang et al. show that the performance improvement due to caching does not translate well to mobile clients with CPU speeds being a bottleneck [17] and many objects on the critical path not being cached [18]. Our paper tries to empirically observe if the limitations still exist with newer phone hardware technologies. Our work differs from Vesnua et al. as i) their work is related to web apps while we exclusively work with PWAs ii) they developed a performance model while we only perform an experiment based study iii) they used page load time as the performance metric while we use CPU, memory and bandwidth.

Google is promoting PWAs through its Workbox [2] library. A case study was done on Google's I/O Web app (IOWA) [19] to analyze the impact of service workers in real-world applications. They divided the service workers into three categories - controlled,

[2]https://developers.google.com/web/tools/workbox

supported(uncontrolled), and unsupported. This study showed that the controlled returning visitors had less load time compared to the new visitors or supported returning visitors. They also observed the time difference between the controlled visitors and uncontrolled visitors for the first pixel rendered on the page is minimal because of idle service worker threads take more time to initialize compared to desktop service workers.

3 EXPERIMENT DEFINITION

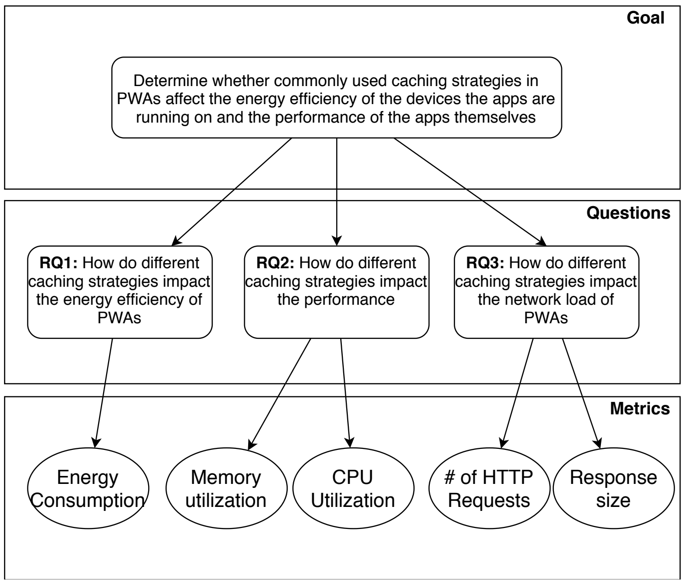

Goal
Determine whether commonly used caching strategies in PWAs affect the energy efficiency of the devices the apps are running on and the performance of the apps themselves
Questions
RQ1: How do different caching strategies impact the energy efficiency of PWAs
RQ2: How do different caching strategies impact the performance
RQ3: How do different caching strategies impact the network load of PWAs
Metrics
Energy Consumption
Memory utilization
CPU Utilization
# of HTTP Requests
Response size

Figure 1: GQM Tree

Analyze	PWA Caching Strategies
For the purpose of	Evaluation
With respect to their	Energy Efficiency, Performance and Network Bandwidth
From the point of view of	Developers
In the context of	PWAs

Table 1: Goal Definition

3.1 GQM Tree

Figure 1 depicts the GQM Tree for the experiment which illustrates the dependencies between the experiment goal, research questions, and the specified metrics.

3.2 Goal

The goal of this experiment is shown in Table 1 and follows the GQM methodology.

3.3 Questions

RQ1 *How do different caching strategies impact the energy efficiency of PWAs?* This question is asked to evaluate whether different caching strategies have an effect on energy consumption based on battery usage.

RQ2 *How do different caching strategies impact the performance of PWAs?* The purpose of this research question is to find out the performance in terms of utilization of the mobile device's CPU and Memory.

RQ3 *How do different caching strategies* affect the network load of PWAs? The purpose of this research question is to evaluate different caching strategies and how they affect the network load size.

Answering the mentioned research questions will help developers understand the strengths and weaknesses of caching strategies in terms of CPU, network performance, and energy consumption. This can help developers choose a strategy for their PWAs. For example, if the app prioritizes network performance over energy efficiency, the developers can select a caching strategy with the best network performance based on our performance comparisons of different strategies.

3.4 Metrics

To effectively answer the formulated research questions, the following metrics will be analyzed:

- Energy Consumption: Battery consumption is measured in Joules.
- CPU Usage: Measured in percentage as utilization.
- Memory Usage: Memory usage is measured in KB.
- Network Bandwidth Size: The size of the HTTP response received in page load in KB/MB
- Number of Network Requests: the number of HTTP requests.

4 EXPERIMENT PLANNING

4.1 Subject Selection

The subjects used for the experiments have been are taken from a combined list of PWAs sourced from PWARocks[3] Awesome-PWA[4] and FindPWA[5]. The 3 sources are community-run and have curated list of PWAs. To have a better comparison between the caching strategies, the PWAs should -

- Either Use a service worker caching library, such as Google Workbox [6] or have a simple and easily modifiable service worker
- Only utilize a single caching strategy

PWAs will also be excluded if -

- Shows the only login and/or register page as a landing page
- Contains broken assets or does load not its components properly on the landing page

[3]https://github.com/pwarocks/pwa.rocks
[4]https://github.com/hemanth/awesome-pwa
[5]https://findpwa.com/
[6]https://developers.google.com/web/tools/workbox

] //the toolbox library called with the cacheFirst strategy toolbox.router.get('/(.*)', global.toolbox.cacheFirst, cache: name: 'googleapis');

To filter out the PWAs to the criteria previously mentioned, the source code of each PWA's service was manually analyzed using Chrome DevTools [7]. The caching strategy was mentioned in the comments or as a library call as shown in listing **??**.

PWA	Library
deanhume.github.io/beer	Toolbox
birdandearth.com.au	Workbox
currency-calc.com	Workbox
m.jetzt.de	Workbox
footballpeek.com	None
devcoffee-pwa.netlify.app	None
resilientwebdesign.com	None
agustinl.github.io/la-cuenta/	None
alexgibson.github.io/wavepad	None

Table 2: List of PWAs used (None signifies simplistic JS service worker codes that only had *fetch* event listeners with simple caching codes and was able to modify fairly easily

From a total of 60 PWAs that were analyzed, 9 PWAs that passed the criteria were chosen as subjects for the experiments (Table 2).

4.2 Experimental Variables

The independent variables will be the **caching strategy** of the PWAs, which will be adjusted for each different run of the experiment The values for the caching strategy are *Cache-First*, *Network-First*, *Network-Only*, *Stale-While-Revalidate*. The *Cache-Only* is not considered as some of the subjects do not work when using said strategy, the reason might be that some of the PWAs require resources to be loaded through the network to load properly.

The dependent variables of this experiment will be **energy efficiency**, performance in **CPU Utilization** and **Memory Utilization** and network bandwidth in **number of requests** and **response size**. The five variables will be monitored starting from the initialization of the PWAs until all the components, including all network responses, have been loaded. The performance variable will consist of the utilization of both the device's CPU and Memory.

4.3 Experimental Hypotheses

The following hypothesis was formulated to address the three research questions -

Given C is the set of available caching strategies, c1, c2 are elements of C and $e(c)$ is the average energy consumption for strategy c for one app, the null and alternate hypothesis are :

$$H_0^e : \forall c1, c2 \in C \;\; e(c1) = e(c2)$$
$$H_a^e : \forall c1, c2 \in C \;\; e(c1) \neq e(c2)$$

Given $p(c)$ is the average CPU performance consumption for strategy c per app, the null and alternate hypothesis are :

$$H_0^p : \forall c1, c2 \in C \;\; p(c1) = p(c2)$$
$$H_a^p : \forall c1, c2 \in C \;\; p(c1) \neq p(c2)$$

Given $m(c)$ is the average memory performance consumption for strategy c per app, the null and alternate hypothesis are :

$$H_0^m : \forall c1, c2 \in C \;\; m(c1) = m(c2)$$
$$H_a^m : \forall c1, c2 \in C \;\; m(c1) \neq m(c2)$$

Given $n(c)$ is the average network bandwidth size metric for strategy c per app, the null and alternate hypothesis are :

$$H_0^n : \forall c1, c2 \in C \;\; n(c1) = n(c2)$$
$$H_a^n : \forall c1, c2 \in C \;\; n(c1) \neq n(c2)$$

Given $r(c)$ is the average number of network requests for strategy c per app, the null and alternate hypothesis is :

$$H_0^r : \forall c1, c2 \in C \;\; r(c1) = r(c2)$$
$$H_a^r : \forall c1, c2 \in C \;\; r(c1) \neq r(c2)$$

The hypotheses are tested for all apps on the device with its Android version and device hardware specified in table **??**.

4.4 Experiment Design

Based on our hypotheses, variables, and the experimental execution for multiple strategies per-app our experiment is designed as One Factor, Greater than Two Treatments. To account for variability in energy consumption, performance, and network bandwidth, each experiment has a repeated measures design, as it is repeated 10 times for each strategy-app combination.

4.5 Data Analysis

The data obtained from the experiment will be analyzed using R scripts to implement the following methods. All tests outlined will be performed considering a significance level α=0.05

- The normality of distribution of the obtained data will be analyzed first using a Q-Q plot to visually test the null hypothesis that normal distribution is present. The Shapiro-Wilk normality test will be performed next to validate the null hypothesis.
- If the null hypothesis that the data follows a normal distribution is validated, an ANOVA test will be run to test each research hypothesis. Alternatively, if the data does not follow a normal distribution, a natural log transformation will be applied to the data to try and get a normal distribution. If this fails, a Kruskal-Wallis test will be performed on the original dataset.
- To measure the magnitude of the effect of caching strategies on the dependent parameters, the non-parametric Cliff's delta test will be performed. This will be done by testing two strategies and then a final time on the two strategies that have the largest magnitude of effect observed from the previous tests.

4.6 Study Replicability

The source code and scripts of this project will be publicly available on Github[8] which can be replicated for independent verification.

[7]https://developers.google.com/web/tools/chrome-devtools

[8]https://github.com/gamma1210/PWACache

The Github repository will contain i) the source code of the PWAs used in the experiment ii) the scripts used for the automation process iii) the resulting dataset of the experiment and iv) the R scripts used for data analysis and visualization.

5 EXPERIMENT EXECUTION

Component	Specifications
Processor	AMD Ryzen 5 3600X (2 cores, 2 threads)
Memory	4002 MB
OS	Ubuntu 20.04 LTS

Table 3: Host Machine Configurations

The experiment is conducted in automated steps executing the PWAs with different caching strategies. The following section describes the implementation procedures with a description of the overall software and hardware infrastructure setup for the experiment. To begin with, about 9 library based service worker PWAs are taken for experimentation like the football peek [9] web app. For automating the experimentation, the tool Android Runner [10] is used. It is a tool for automatically executing measurement-based experiments on native and web application on android devices by making use of plugins, python script injectors using Android Debug Bridge (ADB) commands to communicate with the device based on user-defined configurations. The experiment involves running Android Runner on a host machine with the following configurations as mentioned in Table 3 [11].

Device	Component	Specifications
Huawei Nexus 6P	OS	Android 6
	Chipset	Qualcomm MSM8994 Snapdragon 810 (20 nm)
	CPU	Octa-core (4x1.55 GHz Cortex-A53 & 4x2.0 GHz Cortex-A57)
	GPU	Adreno 430
	RAM	4GB
	Battery	3450 mAh
	Display	1440 x 2560 pixels, 16:9 ratio (518 ppi density)

Table 4: Technical Specifications of Mobile Devices used

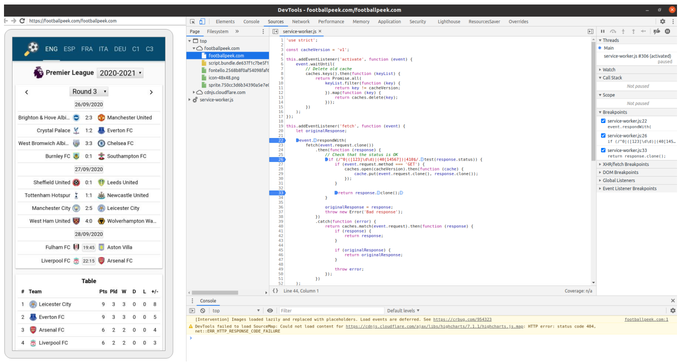
The experimentation is done by running PWAs on an android device. The mobile device configurations are given in Table 4.

PWAs are run using the browser engine contained within the application called the webview. Different browsers have different webviews [20]. The experimentation is planned to be done using the chromium web-engine used in Google Chrome.

[9]https://footballpeek.com/

[10]https://github.com/S2-group/android-runner

[11]Ryzen 5 3600X contains 6 cores and 12 threads, but the experiment is carried out using a virtual machine that is configured to have only 2 cores and 2 threads



```
'use strict';

const cacheVersion = 'v1';

this.addEventListener('activate', function (event) {
    event.waitUntil(
        // Delete old cache
        caches.keys().then(function (keyList) {
            return Promise.all(
                keyList.filter(function (key) {
                    return key != cacheVersion;
                }).map(function (key) {
                    return caches.delete(key);
                }));
        })
    );
});

this.addEventListener('fetch', function (event) {
    let originalResponse;

    event.respondWith(
        fetch(event.request.clone())
            .then(function (response) {
                // Check that the status is OK
                if (/^0|([123]\d\d)|(40[14567])|410$/.test(response.status)) {
                    if (event.request.method === 'GET') {
                        caches.open(cacheVersion).then(function (cache) {
                            cache.put(event.request.clone(), response.clone());
                        });
                    }

                    return response.clone();
                }

                originalResponse = response;
                throw new Error('Bad response');
            })
            .catch(function (error) {
                return caches.match(event.request).then(function (response) {
                    if (response) {
                        return response;
                    }

                    if (originalResponse) {
                        return originalResponse;
                    }

                    throw error;
                });
            })
    );
```

```
[Intervention] Images loaded lazily and replaced with placeholders. Load events are deferred. See https://crbug.com/954323
DevTools failed to load SourceMap: Could not load content for https://cdnjs.cloudflare.com/ajax/libs/highcharts/7.1.1/highcharts.js.map: HTTP error: status code 404, net::ERR_HTTP_RESPONSE_CODE_FAILURE
```

Figure 2: PWA Service Worker debugging using chromium running in a mobile device

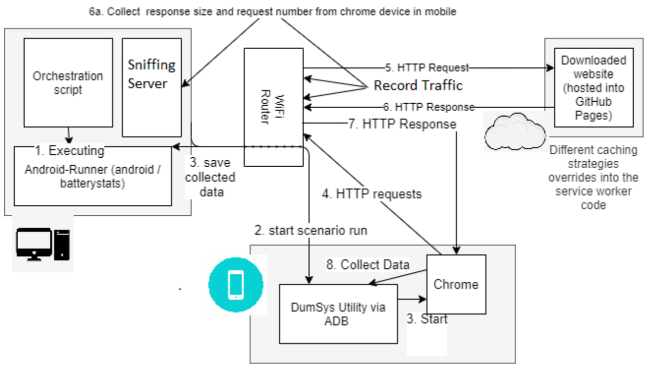

6a. Collect response size and request number from chrome device in mobile
Orchestration script
Sniffing Server
WiFi Router
5. HTTP Request
Record Traffic
6. HTTP Response
7. HTTP Response
Downloaded website (hosted into GitHub Pages)
Different caching strategies overrides into the service worker code
1. Executing Android-Runner (android / batterystats)
3. save collected data
4. HTTP requests
2. start scenario run
8. Collect Data
Chrome
DumSys Utility via ADB
3. Start

Figure 3: Scenario Execution Flow

The mobile device is connected to the host machine through Wi-Fi and uses the respective browser debugging tool via ADB to connect the mobile browser to the host machine's browser as seen in Fig. 2 to override the service worker with different caching strategies or to retrieve the metrics: requests and response size. The PWAs are hosted using GitHub Pages over the network to emulate real-world web app usage scenario, although its made sure only the host and the mobile devices are two components connected to the WiFi network to utilise maximum bandwidth and perform the experiment without any inconsistencies caused by varied bandwidth. Before the experiment can take place, further steps are taken to ensure the setup environment is clean and no noise can affect the results. To ensure this, a clean installation of Android OS with the factory reset, and switching off updates are done. Other applications are blocked from running in the background except for crucial kernel programs and the PWA to be experimented on. Notifications and other background third-party apps are also disabled. Service worker codes are retrieved and made multiple copies of it corresponding to the strategies being experimented on, namely Stale-While-Revalidate, Cache First Then Network, Network First Then Cache and Network Only. These codes are overriden within the GitHub pages to test the corresponding strategy for a PWA.

Using Android-Runner, 9 PWAs are tested upon with each app being made to run for 10 iterations with a duration of analysis of 20 seconds for each run. This configuration is made same to test on all the PWAs to maintain consistency in collecting data. Each app will be loaded and closed from the browser initially before the experiment to load the cache first as populating the cache for the first time can make a highly varied result for the first run in a strategy. About 2 minutes of interval is set up between each run to make sure tail energy usage is taken into account where certain hardware components and threads are kept active preemptively by the kernel to avoid startup energy costs [21].

The experiment involves recording energy consumption, memory utilization, CPU utilization, response sizes and number of HTTP/HTTPS requests being made. These metrics will be measured using the built-in plugins of android runner as well as script injected into the *index.html* page of each PWA. To measure the *energy consumption* of the corresponding PWA, the *batterystats* Profiler tool can be utilised using with the *dumpsys*[12] utility. The energy consumed is measured in *Joules*. Memory and CPU Utilization can be measured using the Android Plugin of Android Runner. Memory will be measure in *Bytes* and CPU utilization in percentage. Response size and HTTP request counts can be measure using the *dumpsys* logger as well as the network monitor of the corresponding webview for the corresponding PWA package. In this experiment, however it was feasible and easy enough to inject a script into the front page of each PWAs to measure these metrics and send it to console logger or to retrieve the data using a dedicated server listening to those metrics from the script.

The experiments are conducted for four out of the five discussed caching strategies. They are namely, *Stale-While-Revalidate*, *Cache First Then Network*, *Network First Then Cache* and *Network Only*. There were several challenges to experiment with the *Cache Only* approach. This experiment is designed to work on established production ready PWAs that are still in use. PWAs are designed to work hand-in-hand with cache and network, and there were very few PWAs that did indeed work with cache only approach. However they failed to satisfy the subject selection criteria as most of them were tiny experimental applications or games that required more interactions. Moreover, the focus on utilising only library based approach in subject selection was inspired by the event that understanding the code for all the PWAs is not a blocking factor and hence a single change in the library based code can function on all the PWAs without errors. The team tasked with the experiments do not need to spend hours in understanding the complex service worker code which if done is outside the scope of this experimentation. Another factor to not include cache only approach is because the offline experimentation of the strategy *Cache First Then Network* is similar to *Cache Only* and the experimentation provided identical results for applications that was storing static HTML pages into cache to provide full offline support. Moreover these applications which stored the entire resources into cache including the HTML pages were only 3 out of the total 9 applications tested out. These blocking factors eliminated the prospect of experimenting on *Cache Only* strategy.

[12]https://developer.android.com/studio/command-line/dumpsys

It was found later during experimental trial execution that PWAs downloaded using website downloader [13] stores static pages and does not have the original PWA functionality of REST API calls to the backend as these requires NodeJS server for the front-end. About 75% of the exhaustive lists of PWAs analysed (around 50-60) for selection were running a front-end server and since it was not open-source, the app selection for the experiment was hard but 9 apps fit in to static PWAs that passed the subject selection. Hence the PWA list was finally listed down to about 9 apps that pass the subject selection as well as is stable in functionality when downloaded without any errors. This is discussed in 8.1.3 as well.

6 RESULTS

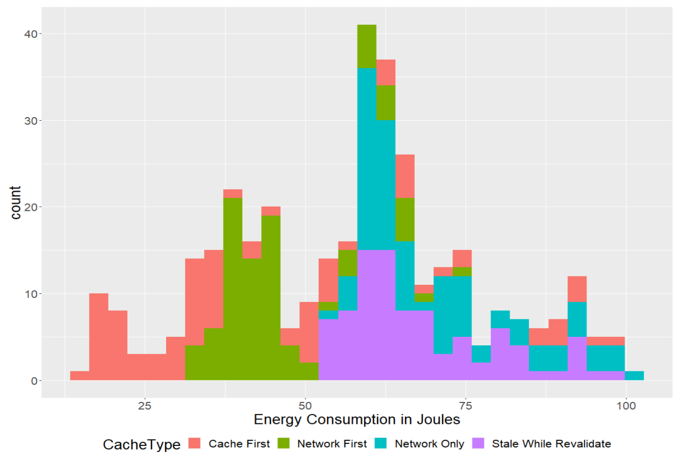


Figure 4: Energy Consumption Histogram

6.1 Impact of different Caching Strategies on Energy Consumption - RQ1

	Energy (J)
Maximum	102.2868
Minimum	15.87456
Mean	57.21746
Median	59.63771
1st Quantile	42.35326
3rd Quantile	68.28103
Standard Deviation	19.47518

Table 5: Energy Consumption

The results of the experiment run for the dependent variable *Energy consumption* are listed in table 5. These metrics have been calculated over the complete dataset obtained after running the experiment for all the apps using all the caching strategies. The minimum energy consumed was observed to be 15.87 J, which

[13]for example: https://www.httrack.com/

was obtained from the *lacuenta* app using the *Cache-First* strategy, while the maximum energy consumed was observed to be 102.28 J, which was obtained from the *WavePad* app using the *Network-Only* strategy respectively whereas the mean energy consumption was observed to be 57.22 J with a standard deviation of 19.48 J. The data distribution for each caching strategy has been depicted as a histogram in Figure 4 which suggests a normal distribution of data by observation of its shape but this will be verified in further steps.

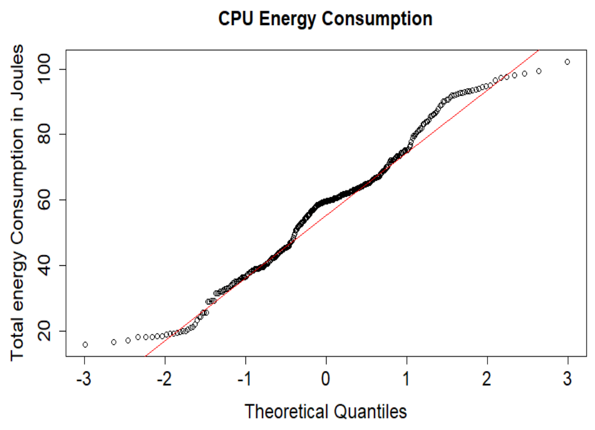


Figure 5: Q-Q plot for energy consumption

The normality of the data distribution has been verified using a Q-Q plot as shown in Figure 5. From direct observation, it can be deduced that the data does not follow a normal distribution. However, to confirm this, performing an additional Shapiro-Wilk test is required which yields a p-value < 2.2e-16 which is quite less than the significance level α=0.05 thereby confirming that the data does not follow a normal distribution.

To check if the data could be made to attain normality, a natural log transformation was applied to the data vector and then another Q-Q plot was generated which is illustrated by Figure 6. This data too did not follow a normal distribution. This was further confirmed by the Shapiro-Wilk test which yielded a p-value = 1.317e-11. Hence, the analysis has been done on the original data to get more accurate results.

To determine if there is a statistically observable difference in the energy consumption by different caching strategies, using the different strategies as four separate treatment factors, a Kruskal-Wallis test was performed because it has been confirmed that the data does not follow a normal distribution. The p-value obtained was 9.293886e-33 which is significantly smaller than the significance level α=0.05. Thus we reject the null hypothesis H_0^e, thereby confirming that different caching strategies do not have the same energy consumption.

Finally, to measure the magnitude of effect each caching strategy has on energy consumption, Cliff's delta test was performed. Cache-First and Network-Only energy readings were tested separately, yielding a medium effect size of -0.6611 in favor of the Network-Only strategy, and Stale-while-revalidate and Network-First energy readings were tested separately, yielding a large effect size of

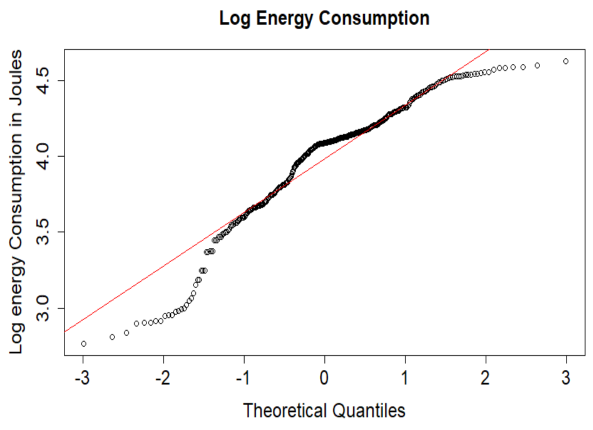


Figure 6: Q-Q plot for log energy consumption

0.8339576 in favor of the Stale-while-revalidate strategy. Finally, Network-Only and Stale-while-revalidate were tested yielding a negligible effect size of 0.08345679 in favor of the Network-Only strategy meaning it has the largest magnitude of the effect.

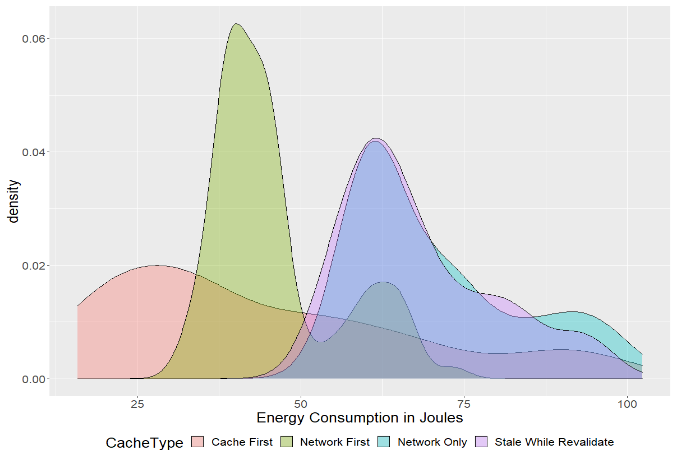


Figure 7: Energy Consumption Density Distribution

To further explore the results of Cliff's delta test, a density plot shown by Figure 7 was generated which shows major overlaps between the Stale-while-revalidate and Network-Only strategies for higher energy consumption readings but the overall spread of energy readings for the Network-Only strategy is higher implying it has the highest magnitude of the effect.

6.2 Impact of different Caching Strategies on the performance of PWAs - RQ2

The results of the experiment run for the dependent variables *Memory utilization* and *CPU utilization* are listed in Table 6. These metrics have been calculated over the complete dataset obtained after running the experiment for all the apps using all the caching strategies. The minimum Memory utilization was observed to be 1073256 bytes, which was obtained from the *lacuenta* app using the *Cache-First* strategy, while the maximum Memory utilization was observed to be 1576331 bytes, which was obtained from the *Dev's Coffee* app using the *Network-Only* strategy respectively whereas the mean Memory utilization was observed to be 1203324 bytes with a standard deviation of 124372.9 bytes. The minimum CPU utilization was observed to be 4.96471%, which was obtained from the *lacuenta* app using the *Cache-First* strategy, while the maximum CPU utilization was observed to be 86.70619%, which was obtained from the *Beer* app using the *Cache-First* strategy respectively whereas the mean CPU utilization was observed to be 20.20874% with a standard deviation of 14.50985% utilization.

	Memory Utilization (bytes)	**CPU Utilization (%)**
Maximum	1576331	86.70619
Minimum	1073256	4.96471
Mean	1203324	20.20874
Median	1177798	15.8074
1st Quantile	1137073	14.01117
3rd Quantile	1196110	19.17457
Standard Deviation	124372.9	14.50985

Table 6: App Performance

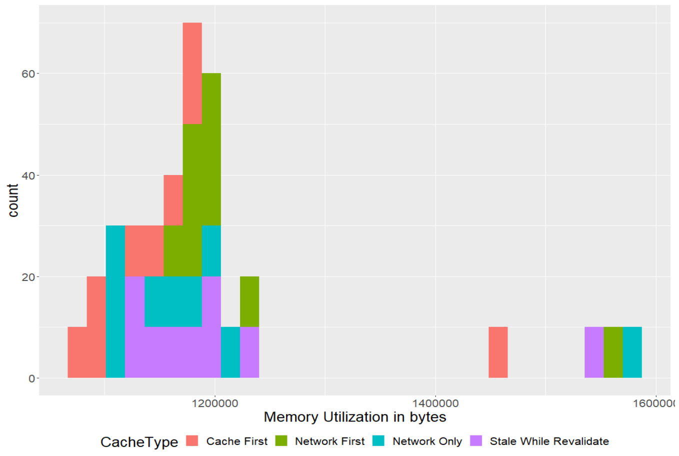


Figure 8: Memory Utilization Histogram

The data distribution for each caching strategy has been depicted as a histogram in Figure 8 for Memory utilization and Figure 9 for CPU utilization, neither of which suggest a normal distribution of data by observation of their shapes but this will be verified in further steps. There are also a number of observable outliers present for both variables. For CPU utilization, they mainly arise from the Cache-First strategy and for Memory utilization, all 4 strategies produce outliers and for different apps.

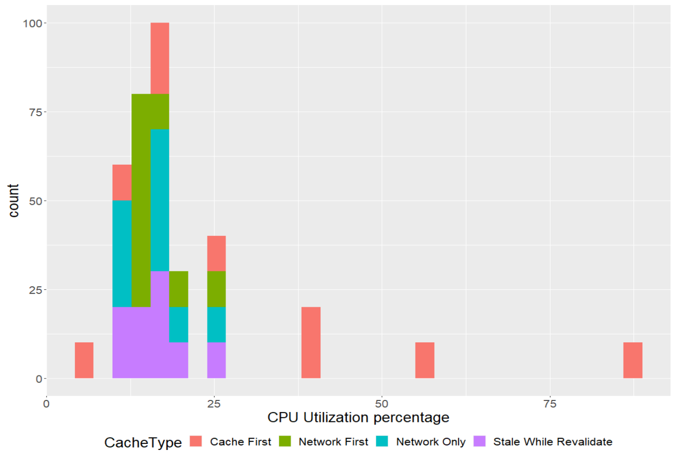


Figure 9: CPU Utilization Histogram

The normality of the data distribution has been verified using Q-Q plots for each variable as shown in Figure 10 for Memory utilization and Figure 11 for CPU utilization. From direct observation, it can be deduced that the data does not follow normal distribution for either variable. However, to confirm this, performing an additional Shapiro-Wilk test is required which yields a p-value < 2.2e-16 for both memory utilization and CPU utilization which is quite less than the significance level α=0.05 thereby confirming that neither data vectors follow a normal distribution.

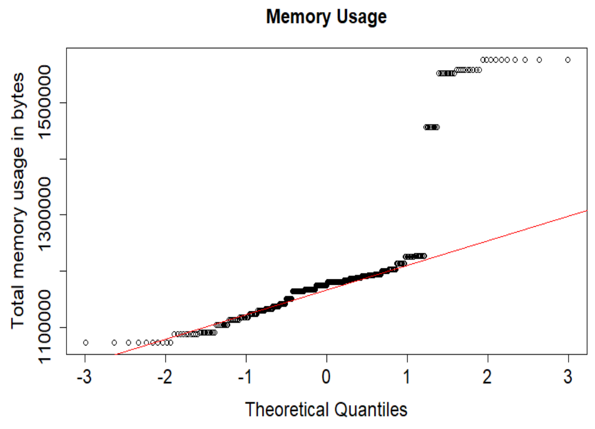


Figure 10: Q-Q plot for Memory utilization

To check if the data for both variables could be made to attain a normal distribution, a natural log transformation was applied to both the data vectors, and then Q-Q plots were generated which are illustrated by Figure 12 and 13 respectively. This data too did not follow a normal distribution. This was further confirmed by the Shapiro-Wilk test which yielded a p-value < 2.2e-16 for both variables. Hence, the analysis has been done on the original data to get more accurate results.

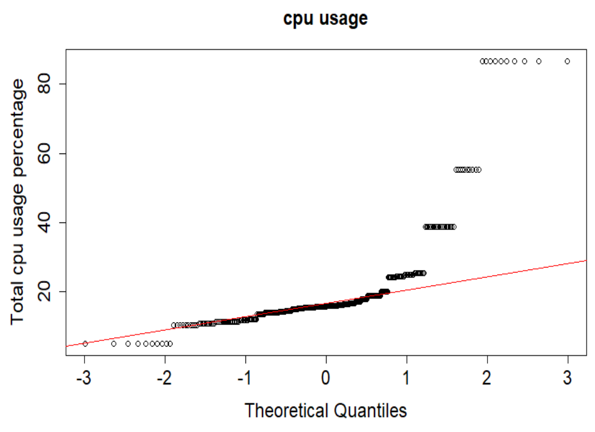


Figure 11: Q-Q plot for CPU utilization

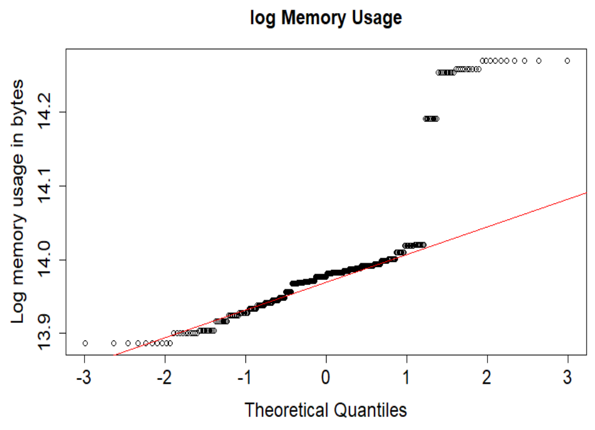


Figure 12: Q-Q plot for log Memory utilization

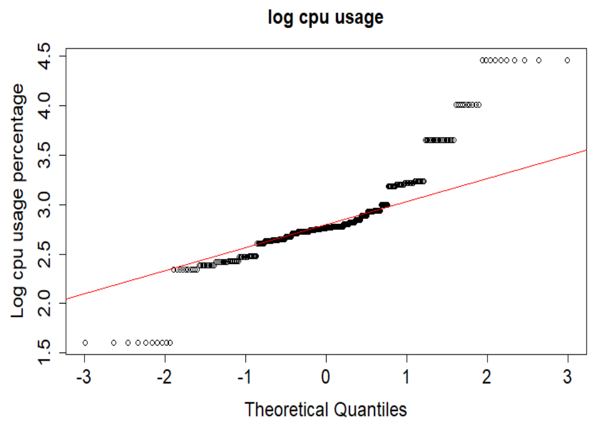


Figure 13: Q-Q plot for log CPU utilization

To determine if there is a statistically observable difference in the Memory utilization and CPU utilization by different caching strategies, using the different strategies as four separate treatment factors, a Kruskal-Wallis test was performed for each variable because it has been confirmed that the data for neither follows a normal distribution. The p-value obtained for memory utilization was 5.702177e-13 and CPU utilization was 2.103712e-07 which is significantly smaller than the significance level α=0.05. Thus we reject the null hypothesis H_0^e, thereby confirming that different caching strategies do not have the same effect on app performance.

Finally, to measure the magnitude of effect each caching strategy has on Memory and CPU utilization, Cliff's delta test was performed for both variables.

- **Memory Utilization**: Cache-First and Network-Only energy readings were tested separately, yielding a small effect size of -0.1990012 in favor of the Network-Only strategy, and Stale-while-revalidate and Network-First energy readings were tested separately, yielding a small effect size of -0.2458025 in favor of the Network-First strategy. Finally, Network-Only and Network-First were tested yielding a medium effect size of -0.360799 in favor of the Network-First strategy meaning it has the largest magnitude of the effect.
- **CPU utilization**: Cache-First and Network-Only energy readings were tested separately, yielding a medium effect size of 0.3495062 in favor of the Cache-First strategy, and Stale-While-Revalidate and Network-First energy readings were tested separately, yielding a negligible effect size of 0.09250936 in favor of the Stale-While-Revalidate strategy. Finally, Cache-First and Stale-While-Revalidate were tested yielding a medium effect size of 0.4017094 in favor of the Cache-First strategy meaning it has the largest magnitude of the effect.

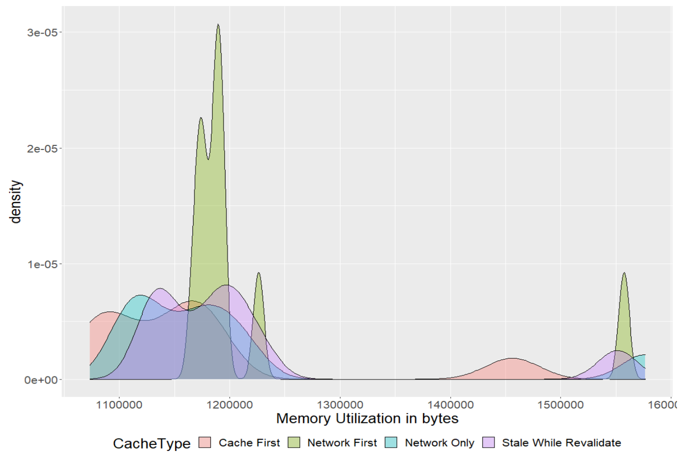


Figure 14: Memory utilization Density Distribution

To further explore the results of Cliff's delta test, density plots were generated for Memory and CPU utilization shown by Figures 14 and 15 respectively. The memory utilization plot shows major overlaps between all the strategies at lower utilizations but significant spread for the Network-First strategy at both high and low

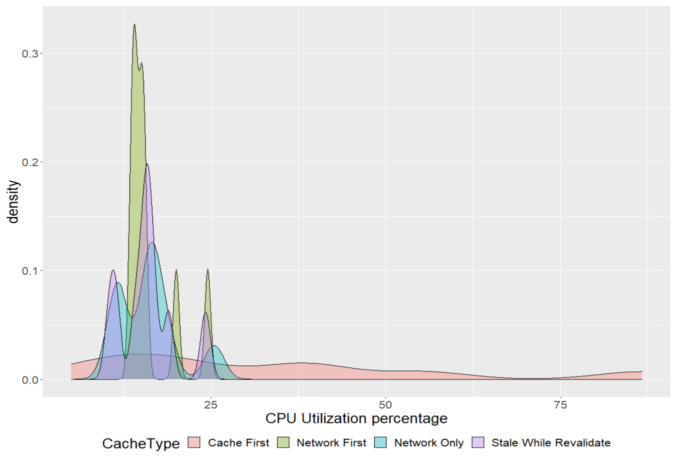


Figure 15: CPU utilization Density Distribution

utilizations meaning it has the highest magnitude of the effect. For CPU utilization, the plot shows overlapping spikes for all strategies except Cache-First at lower utilizations but Cache-First has more variation in spread throughout thus implying that it has the highest magnitude of effect among the 4.

6.3 Impact of different Caching Strategies on PWA network load - RQ3

The results of the experiment run for the dependent variables *number of HTTP requests* and *HTTP response size* required for PWA caching are listed in Table 7. These metrics have been calculated over the complete dataset obtained after running the experiment for all the apps using all the caching strategies. The minimum number of HTTP requests was observed to be 7, which was obtained from the *WavePad* app using the *Cache-First* strategy, while the maximum number of HTTP requests was observed to be 70, which was obtained from the *Bird and Earth* app using the *Stale-While-Revalidate* strategy respectively. The mean number of HTTP requests was observed to be 28.37222 with a standard deviation of 18.86827. The minimum HTTP response size was observed to be 55106 bytes, which was obtained from the *WavePad* app using the *Network-Only* strategy, while the maximum HTTP response size was observed to be 16454114 bytes, which was obtained from the *Dev's Coffee* app using the *Stale-While-Revalidate* strategy respectively. The mean HTTP response size was observed to be 2182806 bytes with a standard deviation of 3826379 bytes.

The data distribution for each caching strategy has been depicted as a histogram in Figure 16 for a number of HTTP requests and 17 for HTTP response size respectively, neither of which suggest a normal distribution of data by observation of their shapes but this will be verified in further steps. There are also a number of observable outliers present for both variables. For CPU utilization, they mainly arise from the Cache-First strategy and for Memory utilization, all 4 strategies produce outliers and for different apps.

The normality of the data distribution has been verified using Q-Q plots for each variable as shown in Figure 18 for the number of HTTP requests and Figure 19 for HTTP response size. From direct observation, it can be deduced that the data does not follow normal distribution for either variable. However, to confirm this, performing an additional Shapiro-Wilk test is required which yields a p-value < 2.2e-16 for both the number of HTTP requests and

	Number of HTTP requests	**HTTP response size (bytes)**
Maximum	70	16454114
Minimum	7	55106
Mean	28.37222	2182806
Median	20.5	806942
1st Quantile	15	428508
3rd Quantile	37	2168862
Standard Deviation	18.86827	3826379

Table 7: Network Load of PWAs

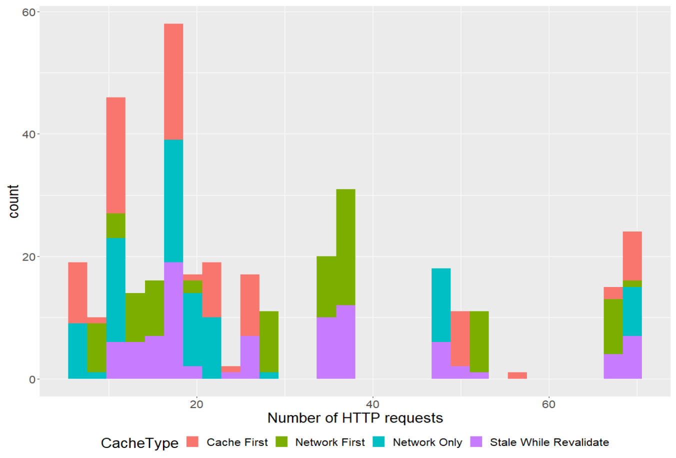


Figure 16: Number of HTTP requests Histogram

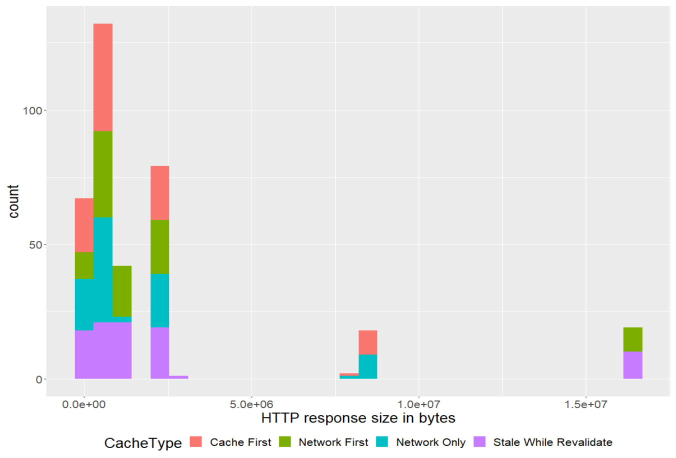


Figure 17: HTTP response Histogram

HTTP response size which is quite less than the significance level α=0.05 thereby confirming that neither data vectors follow a normal distribution.

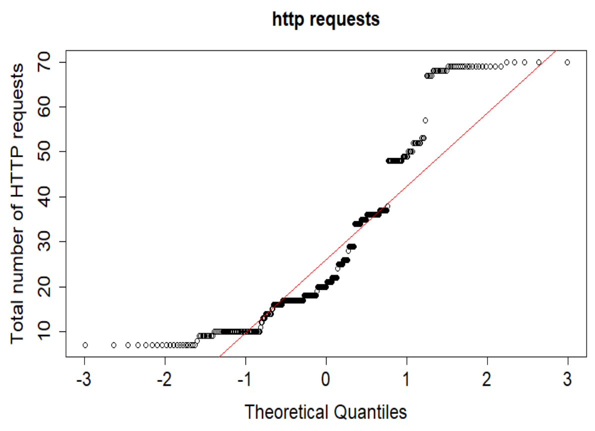


Figure 18: Q-Q plot for number of HTTP requests

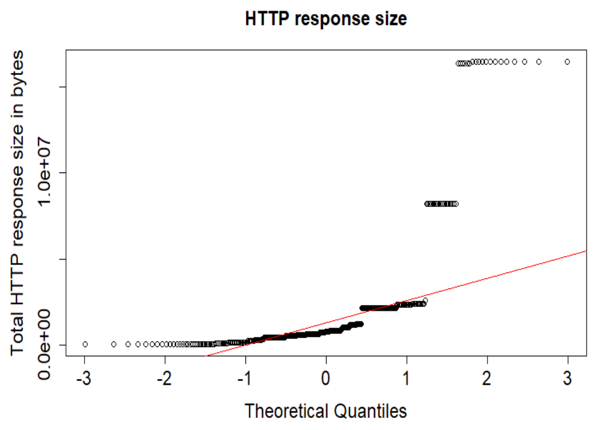


Figure 19: Q-Q plot for HTTP response size

To check if the data for both variables could be made to attain a normal distribution, a natural log transformation was applied to both the data vectors, and then Q-Q plots were generated which are illustrated by Figure 20 and 21 respectively. This data too did not follow a normal distribution. This was further confirmed by the Shapiro-Wilk test which yielded a p-value = 1.662e-09 for the number of HTTP requests variable and p-value = 2.163e-08 for the HTTP response size variable. Hence, the analysis has been done on the original data to get more accurate results.

To determine if there is a statistically observable difference in the number of HTTP requests and HTTP response size by different caching strategies, using the different strategies as four separate treatment factors, a Kruskal-Wallis test was performed for each variable because it has been confirmed that the data for neither

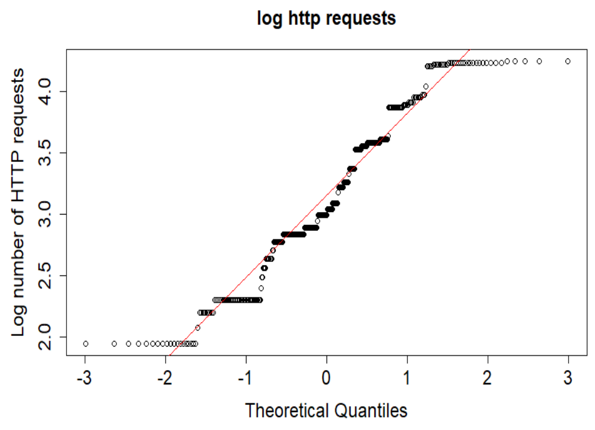


Figure 20: Q-Q plot for log number of HTTP requests

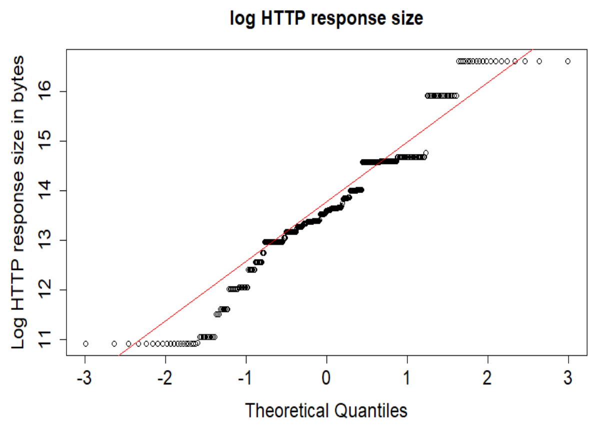


Figure 21: Q-Q plot for log HTTP response size

follows a normal distribution. The p-value obtained for the number of HTTP requests was 0.001358418 and for HTTP response size was 0.03307336 which is significantly smaller than the significance level α=0.05. Thus we reject the null hypothesis H_0^e, thereby confirming that different caching strategies do not have the same effect on the PWA network load.

Finally, to measure the magnitude of effect each caching strategy has on the number of HTTP requests and HTTP response size, Cliff's delta test was performed for both variables.

- **Number of HTTP requests**: Cache-First and Network-Only energy readings were tested separately, yielding a negligible effect size of 0.04839506 in favor of the Cache-First strategy, and Stale-While-Revalidate and Network-First energy readings were tested separately, yielding a negligible effect size of -0.03370787 in favor of the Network-First strategy. Finally, Cache-First and Network-First were tested yielding a small effect size of -0.2636128 in favor of the Network-First strategy meaning it has the largest magnitude of the effect.

- **HTTP response size**: Cache-First and Network-Only energy readings were tested separately, yielding a negligible effect size of 0.02432099 in favor of the Cache-First strategy, and Stale-While-Revalidate and Network-First energy readings were tested separately, yielding a negligible effect size of 0.004619226 in favor of the Stale-While-Revalidate strategy. Finally, Cache-First and Stale-While-Revalidate were tested yielding a small effect size of -0.1506716 in favor of the Stale-While-Revalidate strategy meaning it has the largest magnitude of the effect.

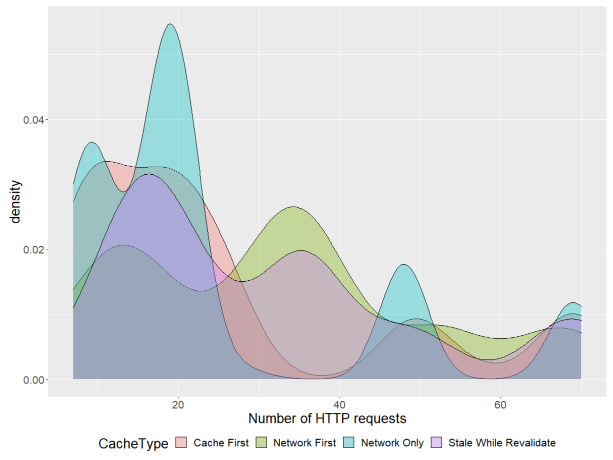


Figure 22: Number of HTTP requests Density Distribution

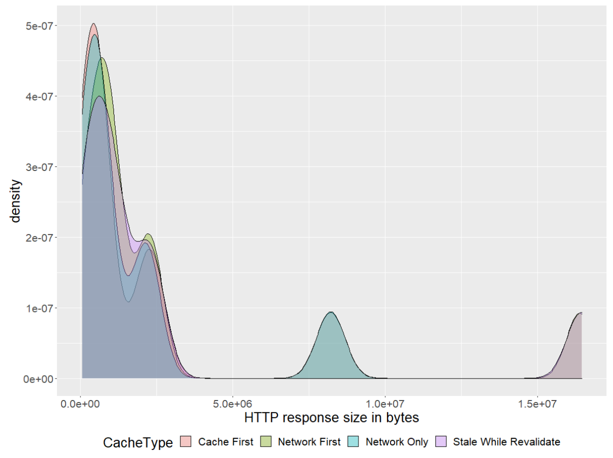


Figure 23: HTTP response size Density Distribution

To further explore the results of the Cliff's delta test, density plots were generated for Number of HTTP requests and HTTP response size shown by Figures 22 and 23 respectively. The Number of HTTP requests plot shows major overlaps between all the strategies and spikes for Network-Only but more variation spread for the Network-First strategy at both high and low utilizations meaning it has the highest magnitude of the effect. For HTTP response size, the plot shows overlapping spikes for all strategies, especially between Cache-First and Stale-While-Revalidate but Stale-While-Revalidate has more variation in spread throughout thus implying that it has the highest magnitude of the effect.

7 DISCUSSION

The results of the experiment were interesting in the sense that certain PWAs are observed to generate consistent performance and utilization metrics for the same caching strategy. An example of this would be the *lacuenta* app which consistently produced low readings for Energy consumption, Memory, and CPU utilization for the Cache-First strategy. This supports the notion that PWA design could also influence its performance metrics along with the caching strategy used.

Furthermore, it was observed that the Network-Only strategy had a greater effect on Energy Consumption, which is reasonable as network operations are usually memory intensive which in turn may be influencing energy consumption. On the other hand, the Cache-First strategy was surprisingly low on energy consumption as it interacts with both the cache and the network (only for a cache miss) and so should have similar energy consumption to Stale-While-Revalidate but had the largest overall effect on CPU utilization due to the same reason.

The Network-First strategy was observed to have the largest magnitude of effect on the number of HTTP requests variable which is reasonable since as per it's the principle, this strategy interacts with the network almost exclusively for caching purposes. For the same reason, the observation that it had the highest Memory utilization is reasonable as network operations are generally memory intensive. Similarly, the Stale-While-Revalidate strategy has the largest magnitude of effect on the HTTP response size variable which is surprising as strategies such as Network-Only or Network-First are logically expected to have more influence on this variable.

To get a better understanding of the dataset, boxplots have been generated for all variables, across all PWAs, and for all caching strategies as illustrated:

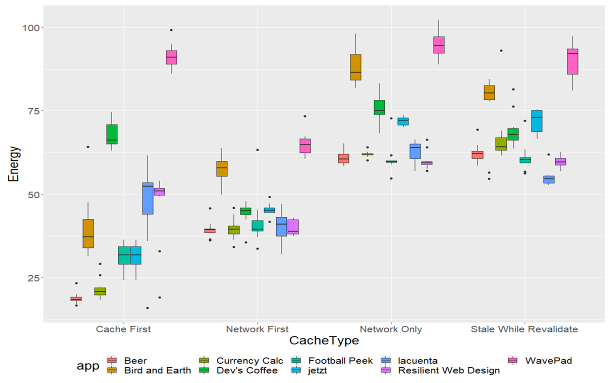


Figure 24: boxplot for Energy consumption per PWA per caching strategy

Figure 24 depicts the energy consumption of each PWA for each caching strategy. The *lacuenta* PWA was observed to have the lowest energy consumption for all strategies except Cache-First and *WavePad* had the highest energy consumption for all strategies.

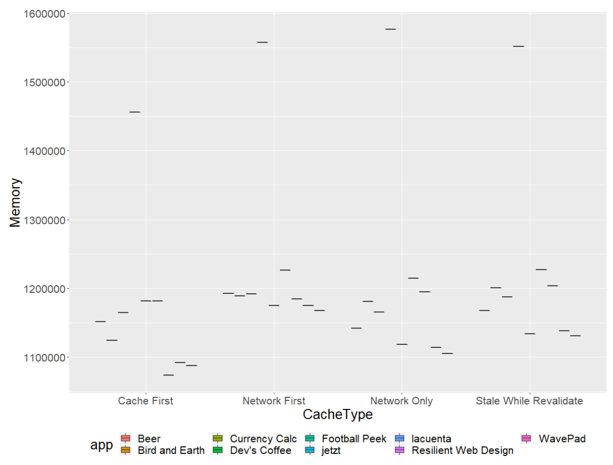


Figure 25: boxplot for Memory utilization per PWA per caching strategy

Figure 25 depicts the Memory utilization of each PWA for each caching strategy. From the box-plot it is deducted that memory utilisation is consistent across iterative runs for PWAs and has almost negligible variance. This shows that this dependent variable does not vary over time and each application utilizes almost a constant memory utilization.

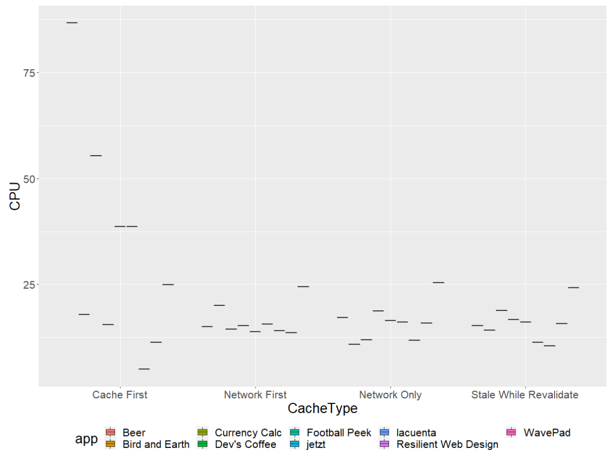


Figure 26: boxplot for CPU consumption per PWA per caching strategy

Figure 26 depicts the CPU utilization of each PWA for each caching strategy. The results are similar to the one discussed for memory utilization. Variances can be seen in memory and cpu if the experiment was conducted without considering initial loading of the PWA. In this case, during the first run, the application has to fetch the results from network and stores the resources into the cache, whereas with the cache already loaded, the application shows consistent utilization of mobile resources over a stable network.

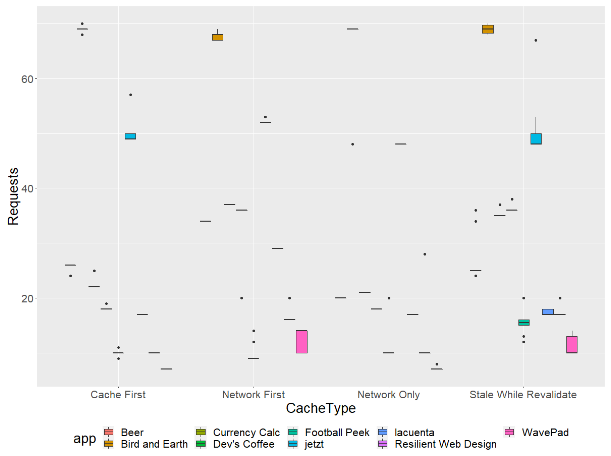


Figure 27: boxplot for Number of HTTP requests per PWA per caching strategy

Figure 27 depicts the number of HTTP requests needed to load each PWA for each caching strategy. The *bird and earth* app has the largest measurements for each caching strategy whereas *WavePad* recorded the overall lowest number of request across strategies. Numerous outliers were also present for all the apps across all strategies but these can be attributed to variation in HTTP calls for cache misses/hits.

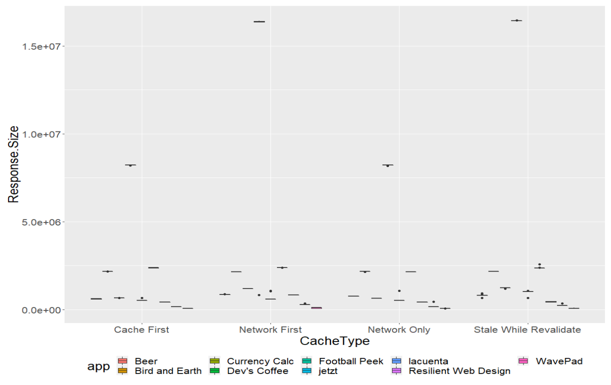


Figure 28: boxplot for HTTP response size per PWA per caching strategy

Figure 28 depicts the HTTP response size of each PWA for each caching strategy. The outliers present can be attributed to variation in HTTP response size when fetching un-cached data over the network. The *Dev's coffee* app has the largest recorded measurements for each caching strategy whereas *WavePad* recorded the overall lowest number of request across strategies.

8 THREATS TO VALIDITY

8.1 Internal Validity

8.1.1 ***History***. The trials were conducted 20 times for each app and the execution was automated by a script to ensure a uniform environment for the results. The manual touchpoints were removed to reduce the effect of this threat.

8.1.2 ***Maturation.*** There is a pause time of 60 seconds before start of each run and the cache is cleared before the start of each run. Separate versions of the code for each cache strategy are used to ensure that there is no caching effect due to previous strategy run.

8.1.3 ***Selection.*** To mitigate this threat, no apps with multistrategy were used for the experiment (which was the case for most of the popular PWAs). Selection plays a role in the experiment as the subjects were selected manually by inspecting their source code and ensuring that there were no errors that caused resources on the landing page to fail, while running the experiment on the subjects hosted on Github Pages[14]. Initially we tried hosting the experiment on a localhost server, however we found that this approach would not be possible as while the service workers loaded and worked in in browsers, when we access the PWA in the device through WiFi, we found that the service workers were unable to be loaded properly. This problem was also present whilst hosting the PWAs through a HTTPS localhost server. The method which we found to work was by hosting the downloaded source of the PWAs on an a live web hosting server, for this experiment the PWAs were hosted on Github Pages. Although this problem was later resolved by installing self-signed certificates into the host browser as well as to the mobile device, we focused our approach to carry out the experiment with pages hosted in GitHub to make the experimentation realistic with adoption of the intrinsic randomness of the network although bandwidth was not changed over time to maintain experimental consistency. The final number of subjects that we were able to get working was 9 PWAs.

It can be said that the apps selected are not a representative sample and our experiment suffers from this threat.

8.1.4 ***Reliability of Measures.*** There are measures like screen brightness, other apps running the background, WiFi connectivity that could affect the results of the experiment. To mitigate this threat, the screen brightness was kept to minimum, a clean install of a device was used which did not contain any other apps and the device was kept in the same position while performing the experiments.

8.2 External Validity

8.2.1 ***Interaction of selection and treatment.*** This threat affects our experiment as all our subjects had a single strategy and the PWAs with a high user count like Flipkart[15], Twitter[16], Tinder[17] all utilise multistrategy, so our selection can't be generalised to all PWAs. But this could not be avoided as we wanted to get results of dependent variables without them getting affected by another strategy in the same execution run. Moreover, most PWAs load some kind of content by doing a request to an API, however we can not consider these PWAs as subjects due to the need of downloading the PWA source and hosting it on our own to make changes to the source code to try out different strategies. If we send a request to the PWAs API we would get a cross origin error due to source of the requests URL (Github Pages).

8.2.2 ***Interaction of setting and treatment.*** This threat is mitigated by hosting the subjects instead of utilizing a local environment. By doing this we are closer to the scenario of an actual, live PWA which is accessed and downloaded through the internet. No other settings of the subjects are disabled to ensure that the settings are not altered as compared to the live app to further mitigate this threat.

8.2.3 ***Interaction of history and treatment.*** This threat does not apply to our experiment as serving the app through Github does not alter the results of the experiment whether we run the experiment on a weekday or a weekend - the results are not affected.

8.3 Construct Validity

8.3.1 ***Inadequate preoperational explication of constructs.*** To mitigate this threat, the constructs were defined using the GQM method before experiment execution which guided the research questions of our study. The independent and dependent variables were also defined during experiment planning to ensure all constructs are defined before experiment execution.

8.3.2 ***Mono-operation bias.*** Since our experiment has one independent variable - the caching strategy, our experiment suffers from Mono-operation bias. To mitigate this threat, we use Greater Than Two Treatments and ran our experiment over multiple repetitions.

8.3.3 ***Mono-method bias.*** To mitigate this threat, two methods are used to check for both Network (Network Bandwidth Size and Number of Network Requests and Performance (CPU, Memory Usage) respectively. For energy efficiency only one method - Energy Consumption in Joules is measured but the experiment is repeated multiple times to reduce the impact the mono method bias, although the experiment still suffers from it.

8.4 Conclusion Validity

8.4.1 ***Low Statistical Power.*** To reduce this threat, we use a sufficient amount of data to work and make conclusions with. Our experiment used a total of 9 PWAs with each having 4 treatments and then ran for 10 iterations each, which brings the total number of experiments to 360.

8.4.2 ***Violated Assumptions Of Statistical Test.*** To mitigate this threat the data was first checked for normality using the Shapiro-Wilk test before applying the appropriate statistical tests. If the resulting data is normally distributed then the ANOVA test would be conducted otherwise the Kruskal-Wallis test will be used. Since, the data was not normal the Kruskal-Wallis test was conducted in our case.

8.4.3 ***Fishing and Error Rate.*** This threat does not apply to our empirical experiment. The tests are not performed repeatedly to look for significant relationships. The presence of outliers and subsequent visualizations prove the absence of fishing for certain results.

[14]https://pages.github.com/
[15]https://www.flipkart.com/
[16]https://twitter.com/
[17]https://tinder.com/

9 CONCLUSIONS

The main focus of this paper was to study the effect of different caching strategies on the Energy consumption, Memory and CPU Utilization, number of HTTP requests and size of the HTTP response necessary for caching for 9 different PWAs. Our results confirm that different caching strategies have different effects on each of the metrics and some strategies have a greater magnitude of effect on particular metrics than the other strategies. This analysis should help PWA developers to decide upon the best choice of caching strategy for their app.

A possible extension of the study would be to study multi-strategy applications and perform the experiment using combinations of strategies to ascertain which combination would perform better in the wild. A second possible extension to the study would be to analyze the Network transfer size and PWA load times as dependant variables because during the course of experimentation, it was observed that these metrics vary wildly from strategy to strategy. A third possible extension would be collaborating with PWA development companies to evaluate the performance of caching strategies for their applications in production which experience high levels of user load instead of downloading their static pages and hosting a version of the application on Github as was done for our study. This in turn could prove useful for formulating a literature study or topic of a thesis for the future.

REFERENCES

[1] P. b. S. O'Dea, "Smartphone users worldwide 2020," Aug 2020. [Online]. Available: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[2] A. Russell, "Progressive Web Apps: Escaping Tabs Without Losing Our Soul," 2020. [Online]. Available: https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/

[3] D. A. Hume, *Progressive web apps.* Manning Publications Co., 2017.

[4] V. I. of TechnologyHead of Technology 5 years. Vitaliy is taking technical ownership of projects including development, "Progressive web apps vs native: How to choose the right mobile app," Aug 2020. [Online]. Available: https://jelvix.com/blog/pwa-vs-native-app-benefits-for-users-and-developers

[5] A. K. Russel, Song, "Service workers w3c candidate recommendation 2019," Nov 2019. [Online]. Available: https://www.w3.org/TR/service-workers/#control-and-use

[6] A. Biørn-Hansen, T. A. Majchrzak, and T.-M. Grønli, "Progressive web apps: The possible web-native unifier," 04 2017.

[7] C. Hoffman, "How to install progressive web apps (pwas) in chrome," Oct 2018. [Online]. Available: https://www.howtogeek.com/fyi/how-to-install-progressive-web-apps-pwas-in-chrome/#:~:text=When%20you're%20on%20a,address%20and%20click%20Install%20%3E%20Spotify.

[8] C. Rojas, "Caching strategies," in *Building Progressive Web Applications with Vue.js.* Springer, 2020, pp. 67–81.

[9] I. Malavolta, K. Chinnappan, L. Jasmontas, S. Gupta, and K. A. K. Soltany, "Evaluating the impact of caching on the energy consumption and performance of progressive web apps."

[10] L. Cruz and R. Abreu, "Performance-based guidelines for energy efficient mobile applications," in *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft).* IEEE, 2017, pp. 46–57.

[11] D. Fortunato and J. Bernardino, "Progressive web apps: An alternative to the native mobile apps," in *2018 13th Iberian Conference on Information Systems and Technologies (CISTI)*, 2018, pp. 1–6.

[12] A. Biørn-Hansen, T. A. Majchrzak, and T.-M. Grønli, "Progressive web apps: The possible web-native unifier for mobile development," in *International Conference on Web Information Systems and Technologies*, vol. 2. SCITEPRESS, 2017, pp. 344–351.

[13] K. Dutta and D. Vandermeer, "Caching to reduce mobile app energy consumption," *ACM Trans. Web*, vol. 12, no. 1, Sep. 2017. [Online]. Available: https://doi.org/10.1145/3125778

[14] I. Malavolta, G. Procaccianti, P. Noorland, and P. Vukmirovic, "Assessing the impact of service workers on the energy efficiency of progressive web apps," in *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, 2017, pp. 35–45.

[15] S. Sivasubramanian, G. Pierre, M. Van Steen, and G. Alonso, "Analysis of caching and replication strategies for web applications," *IEEE Internet Computing*, vol. 11, no. 1, pp. 60–66, 2007.

[16] P. Rodriguez, K. W. Ross, and E. W. Biersack, "Improving the www: caching or multicast?" *Computer Networks and ISDN Systems*, vol. 30, no. 22, pp. 2223 – 2243, 1998. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0169755298002372

[17] J. Vesuna, C. Scott, M. Buettner, M. Piatek, A. Krishnamurthy, and S. Shenker, "Caching doesn't improve mobile web performance (much)," in *2016 USENIX Annual Technical Conference (USENIX ATC 16).* Denver, CO: USENIX Association, Jun. 2016, pp. 159–165. [Online]. Available: https://www.usenix.org/conference/atc16/technical-sessions/presentation/vesuna

[18] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "Demystifying page load performance with wprof," in *Presented as part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)*, 2013, pp. 473–485.

[19] P. Walton, "Measuring the real-world performance impact of service workers," Jul 2016. [Online]. Available: https://developers.google.com/web/showcase/2016/service-worker-perf

[20] T. Steiner, "What is in a web view: An analysis of progressive web app features when the means of web access is not a web browser," in *Companion Proceedings of the The Web Conference 2018*, 2018, pp. 789–796.

[21] D. Li, S. Hao, W. G. Halfond, and R. Govindan, "Calculating source line level energy information for android applications," in *Proceedings of the 2013 International Symposium on Software Testing and Analysis*, 2013, pp. 78–89.