MINT 709, Capstone Project Report

A Quantitative Analysis of Effectiveness of Two Ad Blocking Engines

Prepared by: Mark Leonard

Prepared for: Dr. M. MacGregor

December 2011

Abstract

In the past, the target audience of traditional advertising (such as newspapers, magazines, and billboards) has not incurred costs associated with the ads. This is not necessarily the case when it comes to online advertising. Presumably, online ads increase the total size of webpages that are downloaded - the extra throughput due to the embedded images and other content inherent in advertising. Online ads also represent a potential increase in the time required to load web pages. For companies who pay for Internet usage, increases in page size caused by advertising could have financial consequences.

With some configuration, ad blocking engines can reduce total web page size by approximately 10%. With poor configuration, it is possible to see an increase in total page size due to filtering.

Acknowledgements

I would like to express my sincere gratitude to Dr. M. MacGregor for his guidance and encouragement in the completion of this report. I would also like to thank my parents for their continued support and cooperation throughout my studies. My sisters deserve thanks for their gracious hospitality and accommodations whenever I was in Edmonton for classes and research.

Table of Contents

Table of Figures

1. Introduction

In the past, the target audience of traditional advertising (such as newspapers, magazines, and billboards) has not incurred costs associated with the ads. This is not necessarily the case when it comes to online advertising. Presumably, online ads increase the total size of webpages that are downloaded - the extra throughput due to the embedded images and other content inherent in advertising. Online ads also represent a potential increase in the time required to load web pages. For companies who pay for Internet usage, increases in page size caused by advertising could have financial consequences.

This project has the following aims:

1) The primary aim is to determine if blocking advertisements can decrease the amount of data transferred when viewing a webpage
2) If the primary aim is met, a secondary aim is to determine if ad blocking can be performed without negatively impacting page load times

1.1. Types of content used by advertisers

Online advertisements are not strictly images. In fact, there are a number of different technologies that are used to both display ads and ensure that accurate metrics are being returned about the number of ads that have been viewed. These technologies are broken down into two major categories:

1) Advertisements - These can take the form of images, text, animations, videos, audio, in various combinations. Ads are designed to draw attention and can occupy much of the viewable area on a webpage.
2) "Web bugs" - The advertisers want to know that the ad-space they are paying for is getting a good return on their investment. To this end, advertisers and marketers inject webpage elements like javascript, dynamically generated images, and other content into a webpage. These elements are designed to report metrics such as how long users stayed on a site, what they were doing, and whether or not the advertisements were displayed. They do not represent a significant impact in terms of bandwidth used when viewing a site, but they could represent a significant impact in terms of page load times.

1.2. Factors affecting page load times

The amount of time it takes to load a webpage is dependent on several factors such as:

1) The round-trip time for data connectivity to a server
2) The load on the server (this is related to the number of people visiting the page simultaneously)
3) The available bandwidth on the path between the user and the server
4) The nature of the requested data (static content versus dynamic)
5) The number of elements on a webpage

6) The server hardware (modern hardware can normally serve webpages faster than legacy hardware)

1.3. Technologies for filtering advertisements and web bugs

Adblock Plus[1] - This is a plug-in available for Mozilla Firefox and Google Crome. Adblock Plus is often configured to subscribe to a free list of blockable content. The strength of this solution is the automatic updates that are pushed via the subscribed list. The weakness is that it must be configured on each computer and possibly for each user account on each computer.

IE7Pro[2] - This is an add-on available for some versions of Internet Explorer that supports content filtering such as Ad Blocking. From what little I have been able to find online about this tool, it is not as refined as Adblock Plus. The default rules are quite broad and will likely block desirable content. It is configurable for more fine-grained filtering, but this is time consuming for end-users. IE7Pro is not available for all computer platforms as it was designed to work exclusively with the Microsoft Windows™ environment.

Privoxy[3] - Privoxy is a web proxy that can filter and does not cache data. It is designed for both single-user computers and multi-user networks. Privoxy is capable of more than just blocking - it can manipulate cookies and other content as it proxies the data. It does not appear that there is any built-in method to subscribe to blocklists. Privoxy still requires that each computer on a network is setup to use Privoxy as a proxy.

Privoxy-blocklist[4] - this is a script to translate the subscribeable blocklists from Adblock Plus and use them in Privoxy. This combines the power of Adblock Plus' continually updated lists with the increased functionality of Privoxy.

In this paper Adblock Plus and Privoxy-blocklist will be compared. Both of these can be scripted from Linux and thus controlled for repeated test execution.

1.4. Problems benchmarking a website

Website design is becoming increasingly complex. In the past webpages referenced static images and content. There was a clear and definitive end to mark when a webpage had completed loading. Today flash, javascript and other dynamic web tools allow more content to be downloaded after the initial webpage is transferred. This download process can be indefinite.

[1] http://adblockplus.org/en/
[2] http://www.ie7pro.com/
[3] http://www.privoxy.org/
[4] http://andrwe.org/doku.php/scripting/bash/privoxy-blocklist

The site http://www.nytimes.com offers a good example of a website that never finishes loading. Periodically, additional web bugs are downloaded to (presumably) give the site maintainers an indication of how long a particular user is looking at a page. Alternatively, if the visible content changes, this could be used to keep the viewer engaged.

To work around the problem of perpetually loading webpages I established a cut-off period. If there is no network traffic to or from a website for a predefined period[5], I consider the site fully loaded. Given the nature of retry intervals in the TCP/IP stack, it is entirely possible that a site with no network activity during this period is still loading. This is a limitation of this testing methodology. The predefined period was arbitrarily chosen by me as an unreasonable amount of time to wait for nothing to happen - I assume that future generations accessing website are going to be impatient as or more impatient than I am.

1.5. Site Selection

Alexa[6] publishes a daily list of the top one-million most popular websites according to their statistics. Their list suffers from opt-in selection bias. To generate their statistics, Alexa publishes a browser toolbar that monitors and reports a user's activity back to Alexa. If users do not install the toolbar, their web browsing statistics are not known by Alexa.

For these tests, the list of the top one-million sites was downloaded from Alexa on 12 November 2011. The top 200 sites were extracted from this list and then filtered. Filtering removed any site with "porn" in the name - these sites are likely discouraged from use in most corporate environments. The list was sorted alphabetically so that duplicate sites with different country-code top-level domain names were shown (like google.ca in addition to google.com). The duplicates were removed. The filtered list consists of 163 sites.

1.6. Testing criteria

To test our hypothesis, a pair of tests must be completed with the following criteria followed:

1) Close to each other (in terms of time) - Different websites have different usage patterns. These usage patterns are not generally disclosed to the public. If the first test in a pair happened when the site usage was significantly different from the second test in the same pair the results would be invalid. It is also possible that site content will change throughout a day, but it is less likely that there will be a significant change within a shorter period of time. For these reasons, the two tests in a pair must be chronologically close to each other.
2) On the same network topology on the WAN - The networking must be as stable as possible during the tests. On the WAN, dynamic routing protocols take time to adjust to changes - this is a

[5] This is set by the parameter MAXDELTA in Appendix D - /root/benchmark.py
[6] http://www.alexa.com/

limitation of the BGP routing protocol. If the pair of tests are chronologically close to each other this requirement is considered to be followed as well as possible. There could be some outliers in the results due to WAN changes that occur between two tests in a pair.

3) Consistent usage on the LAN - Changes in the LAN are easily avoided by not making changes to the infrastructure. It is also important to ensure that network utilization on the LAN remains constant. It is possible that heavy network utilization on the LAN could lead to contaminated data. A sudden spike in local network traffic could lead to contention for bandwidth and negatively affect the apparent page load times. It's also possible that too much contention for bandwidth could cause data loss, leading to the scripts prematurely identifying the site as fully loaded due to a one-second interval without appropriate site traffic.
4) On the same hardware - needless to say if the hardware running the tests is different for the two items in the pair it could cause results to be skewed.
5) With the same software configuration - the OS version and the browser version could affect the manner in which elements of the website are loaded. Keeping these constant for a set of statistics is imperative.
6) Avoid being labeled as an attacker - If too many requests for the same data happen too quickly, the site administrators may view the tests as a Denial of Service (DoS) attack upon the site and block the test script's access to the site. Sufficient time must be allowed between requests so that the requests the script is initiating, blend in with other site traffic. There is no sure-fire way to accomplish this. The best approach I could come up with was to space out the requests to reduce the chance of detection.

Variations due to the above criteria are unavoidable. A large enough sample should cause variations to follow a normal distribution.

2. Methodology

Multiple test pairs will need to be completed in order to collect sufficient samples to be able to have statistically significant results. Given the large number of sites being tested, a single set of test pairs for each site takes just over 1.5 hours to collect. The tests were scheduled to run on two-hour intervals to allow for some buffer if the tests took longer than expected:

```
tpb@webtest:~$ sudo crontab -l
0 */2 * * * /root/test.sh
```

The scripts used for benchmarking are included in the appendices.

The main script, "test.sh" performs the following:

1) Clears Firefox's offline cache - The cache stores images and other objects that may have been downloaded previously and are still considered usable. This content must be deleted to ensure that each test downloads a complete webpage with all of its included assets.

2) Clears Firefox's tab session data - Firefox is prone to re-open recent tabs when shutdown abruptly (as in a scripted shutdown). The next time Firefox is launched it will attempt to pickup where it left off plus any new sites that are being requested. This session data must be deleted to ensure that only the target site is being requested.
3) Enable or disable ad filtering.
4) Spawns a script (/root/webpage-bench.sh and /root/benchmark.py) to record data via TCPDump to track the time and traffic used to download the site. The second script (benchmark.py) is used to detect when the webpage has finished loading and to terminate the Firefox client as well as the TCPDump session then write the gathered statistics to a file.
5) Runs a wrapper script for Firefox. The wrapper is required to set the appropriate environment variables to ensure that Firefox has a display that it can attach to.
6) There are 5-second gaps between each test. This is to ensure no overlap in TCPDump data nor any other cross-contamination between the tests.

As pseudocode, the sequence above is simplified:

```
for each site in list:
      {
      enable ads
      benchmark unfiltered site
      wait 5 seconds
      disable ads
      benchmark filtered site
      wait 5 seconds
      }
```

The full script can be found in Appendix A.

In order to meet the testing criteria of having the two tests (filtered and unfiltered) for a site within a short period of time, the filtered test was run 5 seconds after the unfiltered test for the same site. The algorithm (in pseudo-code) is:

With tests run on 2-hour intervals, the odds of being labelled as an attacker were diminished.

For all tests, Firefox was configured to not check for updates automatically (this could interfere with results).

2.1. Sample validity

The criteria for determining sample validity are:

- Bytes transferred must not be zero
- Transfer time must not be zero
- The change in page size must be a factor between 1.8 and 0.2

A page size of zero, or a page load time of zero would indicate that the page was not loaded at all. A change in page size of almost 80% would indicate that the page source has changed, invalidating the sample pair. Any of these problems would indicate that the sample pair is invalid and should be excluded from the statistics.

2.2. Test Overview

The following three tests were conducted:

- Test #1 – Individual workstation using Adblock Plus
- Test #2 – Privoxy-blocklist with minimal changes to configuration
- Test #3 – Privoxy-blocklist with a modified configuration

3. Test #1 – Individual workstation using Adblock Plus

3.1. Description

For this test, the filtering will be enabled on the client computer itself with Firefox's add-on module Adblock Plus. The purpose of this test is to establish if filtering at the web browser is capable of reducing throughput and page load time. This type of filtering must be configured on each workstation and may not be cost effective for larger organizations. This test will also serve as a baseline for expectations of other tests.

3.2. Methodology specific to test #1

Adblock Plus was enabled and disabled by overwriting the configuration files for Firefox with previously stored configurations that had Adblock Plus enabled or disabled, as discussed in the Methodology section of this document.

This test was run for several days to accumulate a large sample size.

3.3. Results

The primary objective was to see if there was a change in the total size of the webpages. This is the scatter plot comparing the values for the test pairs:

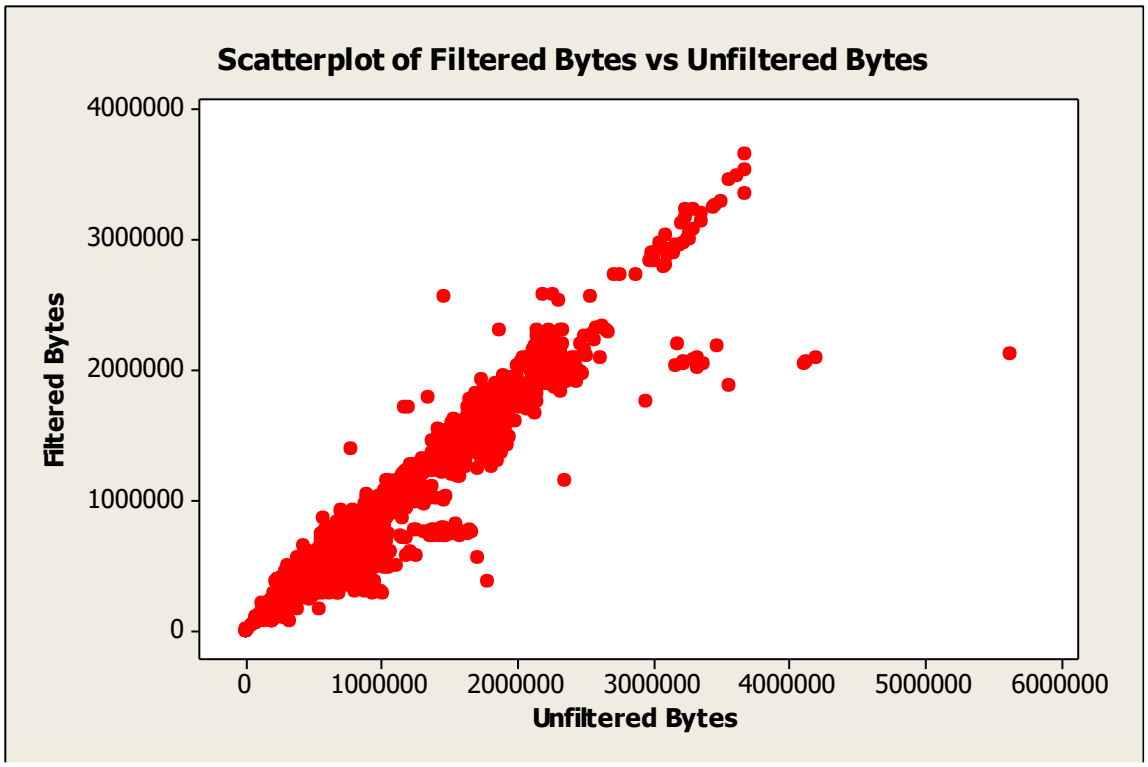

Figure 1: Test #1, Scatterplot of Filtered Bytes vs Unfiltered Bytes

This graph indicates a linear or near-linear relationship between filtered bytes and unfiltered bytes.

There are several outliers. These outliers are statistically insignificant due to the sample size. They are within the bounds of sample validity and are thus included.

Although a linear relationship may be present, ad filtering effectively decreases page size not by a percentage, but instead by the size of the advertizing content. As such, I have opted to examine the delta in size and time between unfiltered and filtered pages rather than the ratio of filtered to unfiltered.

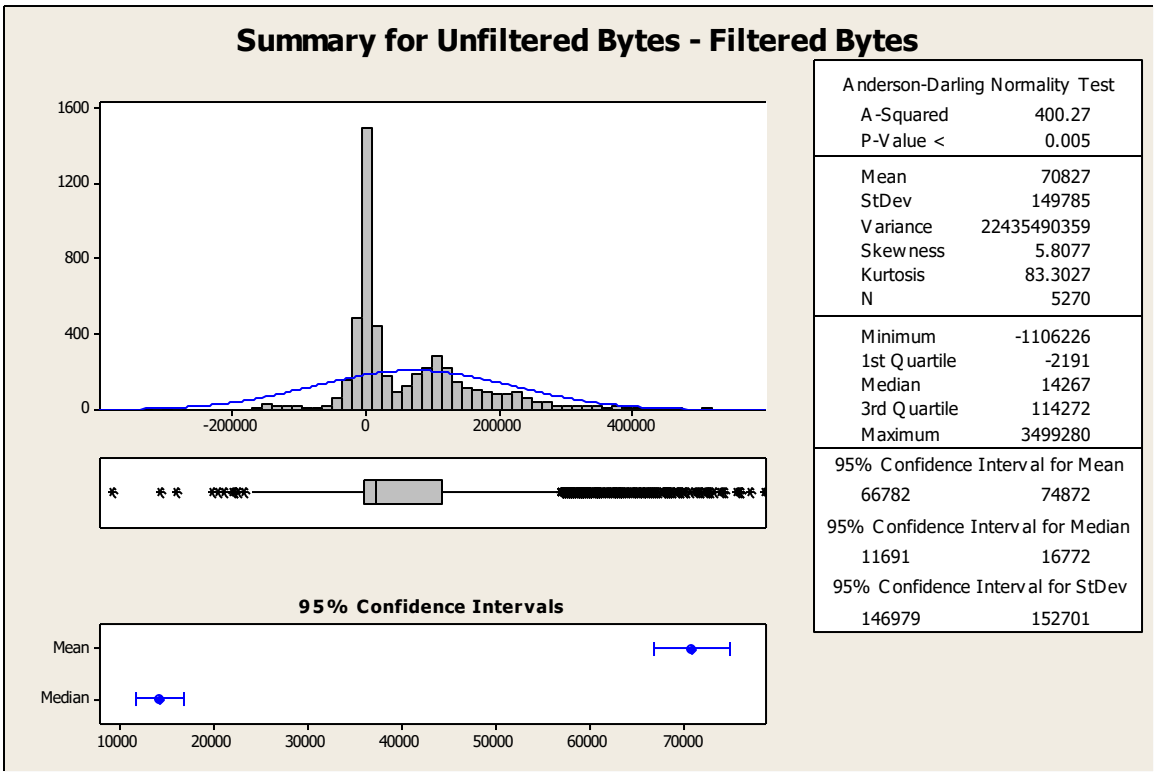

Figure 2: Test #1, Summary for Unfiltered Bytes - Filtered Bytes

The Histogram of Filtered Bytes minus Unfiltered Bytes (in the top left of Figure 1) shows how often there is a decrease in page size due to filtering (any value above 0) and how often there is an increase in page size due to filtering (any value below 0). It is interesting that this histogram is bi-modal, which could indicate a factor that characterizes the data distinctly; however this is not the focus of our investigation for this test. This histogram indicates that the majority of the pages saw a decrease in page size due to filtering. The mean and the median also indicate that there is a decrease in page size. With the hypothesis that there is a decrease in page size, we can build the following test of hypothesis:

H_0: $u = u_0 = 0$ there is no change in mean webpage size with filtering
H_1: $u > u_0 = 0$ there is a decrease in mean webpage size with filtering

The paired z-Score was calculated with:

$$z = \frac{\overline{x_r} - u_0}{s_r / \sqrt{n}}$$

Where $\overline{x_r}$ is the sample mean of the paired differences of Filtered Bytes minus Unfiltered Bytes in pairs.

Where s_r is the sample standard deviation of the paired differences of Filtered Bytes minus Unfiltered Bytes in pairs.

Rejection region for these tests will be set at $z > 3$. This will provide us with over 99% confidence.

In this case, $z=34.3$ and the p-value < 0.001. This is sufficient to reject the null hypothesis.

This indicates that this is a net decrease in page size due to filtering traffic with this method.

The next hypothesis that must be examined is whether or not there is a decrease in page load times due to filtering. This is a secondary requirement for any ad-filtering as described earlier in this paper.

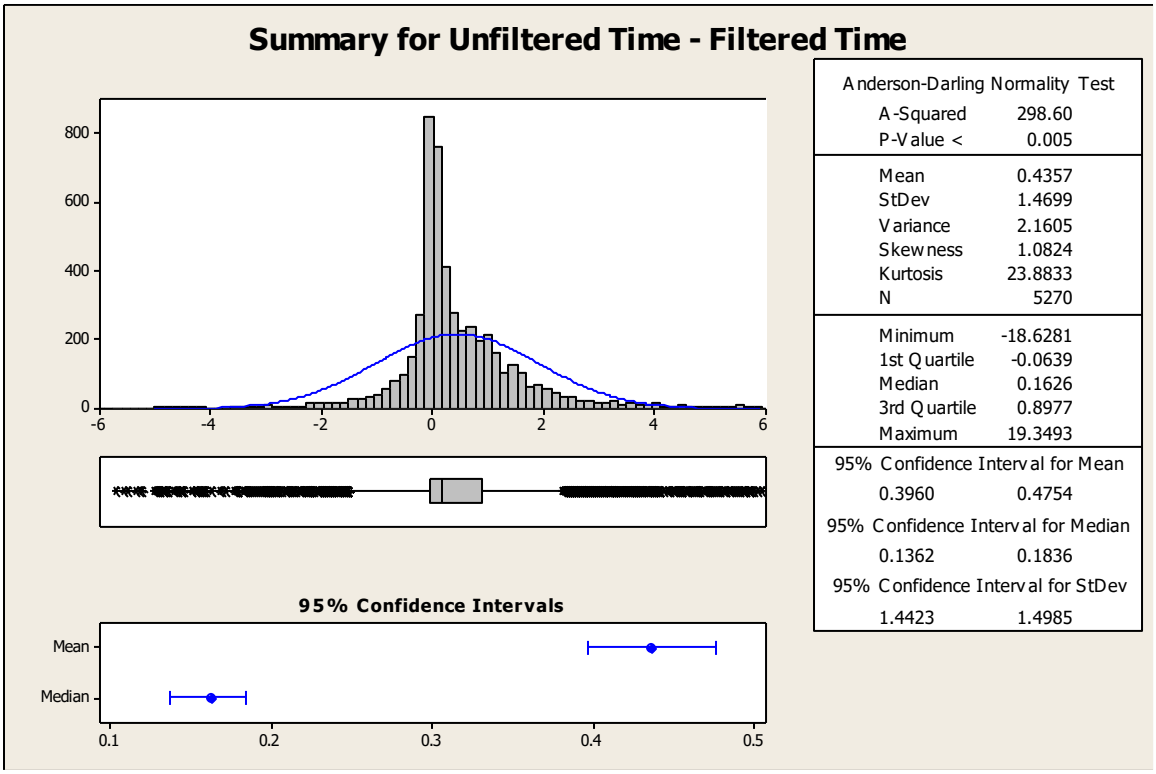

Figure 3: Test #1, Summary for Unfiltered Time - Filtered Time

From the histogram in the top left of this chart as well as the median and the mean, it appears as though there is a decrease in page load time due to filtering. We can proceed with this test of hypothesis:

H_0: $u = u_0 = 0$ there is no change in mean webpage load time with filtering
H_1: $u > u_0 = 0$ there is a decrease in mean webpage load time with filtering

The paired z-Score was calculated with:

$$z = \frac{\bar{x}_r - u_0}{s_r / \sqrt{n}}$$

Where $\overline{x_r}$ is the sample mean of the paired differences of Filtered Time minus Unfiltered Time (in pairs).

Where s_r is the sample standard deviation of the paired differences of Filtered Time minus Unfiltered Time.

Rejection region for these tests will be set at $z > 3$. This will provide us with over 99% confidence.

In this case $z=21.5$ and p-value < 0.001. There is sufficient evidence to reject the null hypothesis.

This indicates that with filtering, the page load time will decrease.

3.4. Conclusions for test #1

Using Adblock Plus in Firefox will reduce both page size and page load times. This conclusion is intuitive as there is less data being received by the workstation.

4. Test #2 – Privoxy-blocklist with minimal changes to configuration

4.1. Description

For the second test, an intercepting proxy was configured on the LAN to intercept and replace traffic from client computers listed in an OpenBSD Packet Filter (PF) table. The client machine is dynamically added and removed from the PF table in order to turn on or off filtering. This form of filtering and proxying can be applied to an entire network, rather than each machine individually. This form of filtering would be most useful for medium to large organizations where it is more cost efficient to have all filtering done in one location rather than at each individual workstation.

4.2. Methodology specific to test #2

The default configurations were used as much as possible. It was necessary to configure Privoxy to function in "intercept" mode as the Firefox client was not explicitly configured to use a proxy.

In /etc/privoxy/config:

edited line 1366: `accept-intercepted-requests 1`
edited line 735: `listen-address 192.168.1.2:8118`

These changes should have no impact on the test results because there are no changes to how the filtering engine parses its rules.

To enable and disable ad filtering the test.sh script (Appendix A) would remotely execute a "pfctl" command on the gateway to add or remove the host from the list of hosts that ad filtering is applied to.

The proxy is always configured to filter ads based on a downloadable blocklist (the same blocklist used in Test #1 with Adblock plus).

To have OpenBSD's PF proxy the information and adjust the source IP as required, the following configuration is required[7]:

```
table <adfree> persist
pass in quick on $int_if proto tcp from <adfree> to any port 80 rdr-to 192.168.1.2 port 8118
pass out on $int_if proto tcp to 192.168.1.2 port 8118 received-on $int_if nat-to $int_if
```

4.3. Results

As the primary objective was to see if there was a change in the total size of the webpages, I first looked at the scatter plot comparing the values for the test pairs.

[7] http://openbsd.org/faq/pf/rdr.html#rdrnat

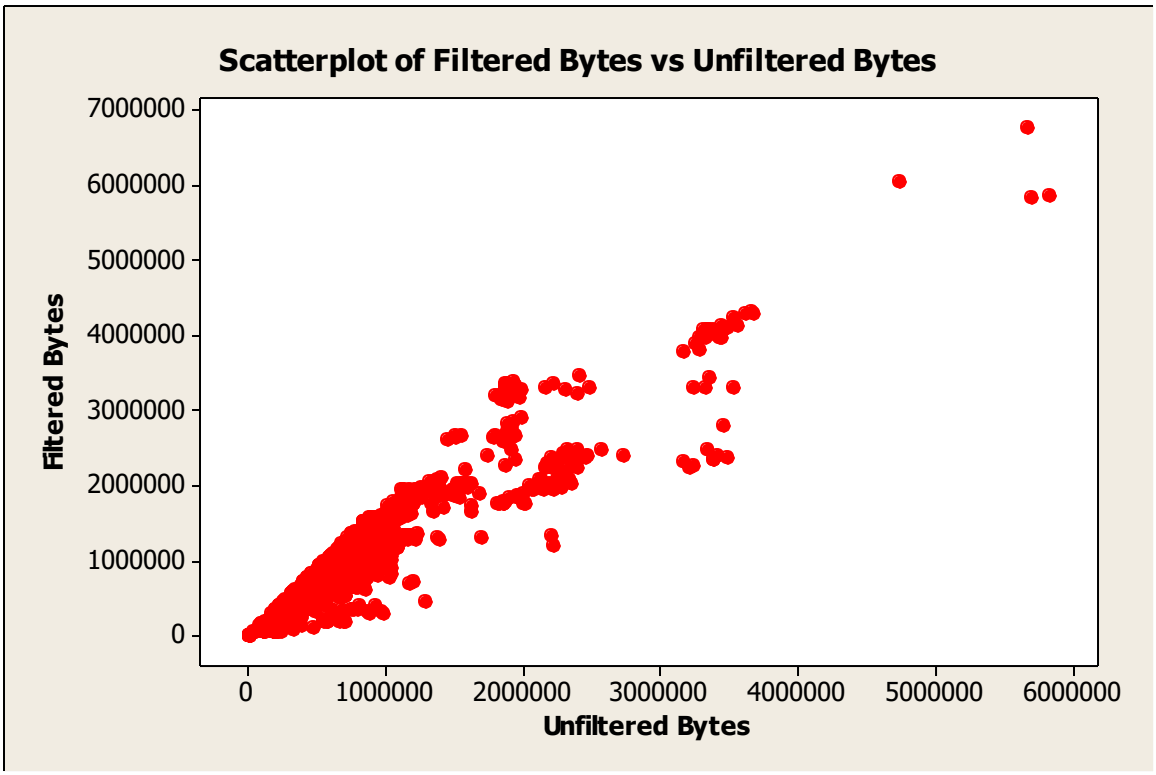

Figure 4: Test #2, Scatterplot of Filtered Bytes vs Unfiltered Bytes

This scatterplot appears to be roughly linear, as in the first test. Again I will analyze the delta between Unfiltered Bytes and Filtered Bytes:

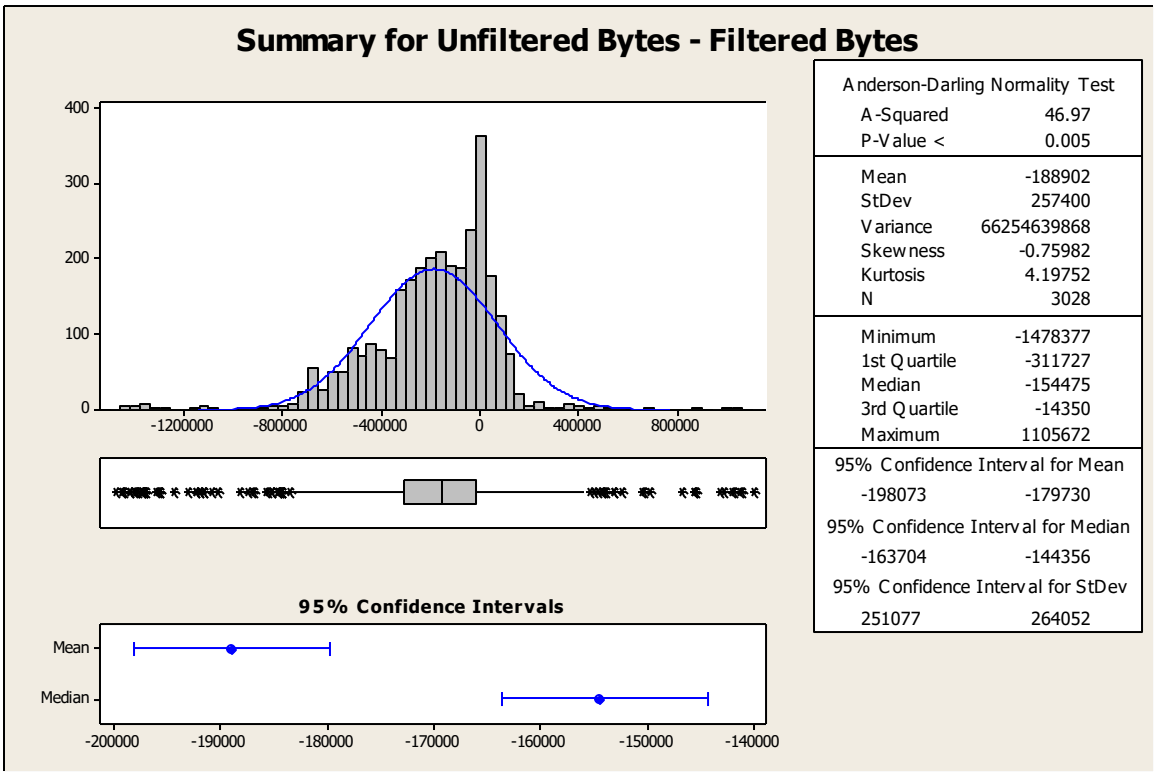

Figure 5: Test #2, Summary for Unfiltered Bytes - Filtered Bytes

The histogram of the difference between unfiltered and filtered bytes in the top left of Figure 5 has a large volume of datapoints below zero. The mean and median are also below zero. This indicates that it is more likely that there was an **increase** in the size of webpages when filtering was enabled versus disabled. To verify this new hypothesis, we setup the following test:

H_0: $u = u_0=0$ there is no change in mean webpage size with filtering

H_1: $u < u_0=0$ there is an **increase** in mean webpage size with filtering

The paired z-Score was calculated with:

$$z = \frac{\bar{x}_r - u_0}{s_r / \sqrt{n}}$$

Where $\overline{x_r}$ is the sample mean of the paired differences of Filtered Bytes minus Unfiltered Bytes.

Where s_r is the sample standard deviation of the paired differences of Filtered Bytes minus Unfiltered Bytes.

Rejection region for these tests will be set at $z > 3$. This will provide us with over 99% confidence.

In this case, z=-40.4 and the p-value < 0.001. This is sufficient evidence to reject the null hypothesis.

Although our primary hypothesis was not met (a decrease in page size), there may still be some value in examining the change in page load time:

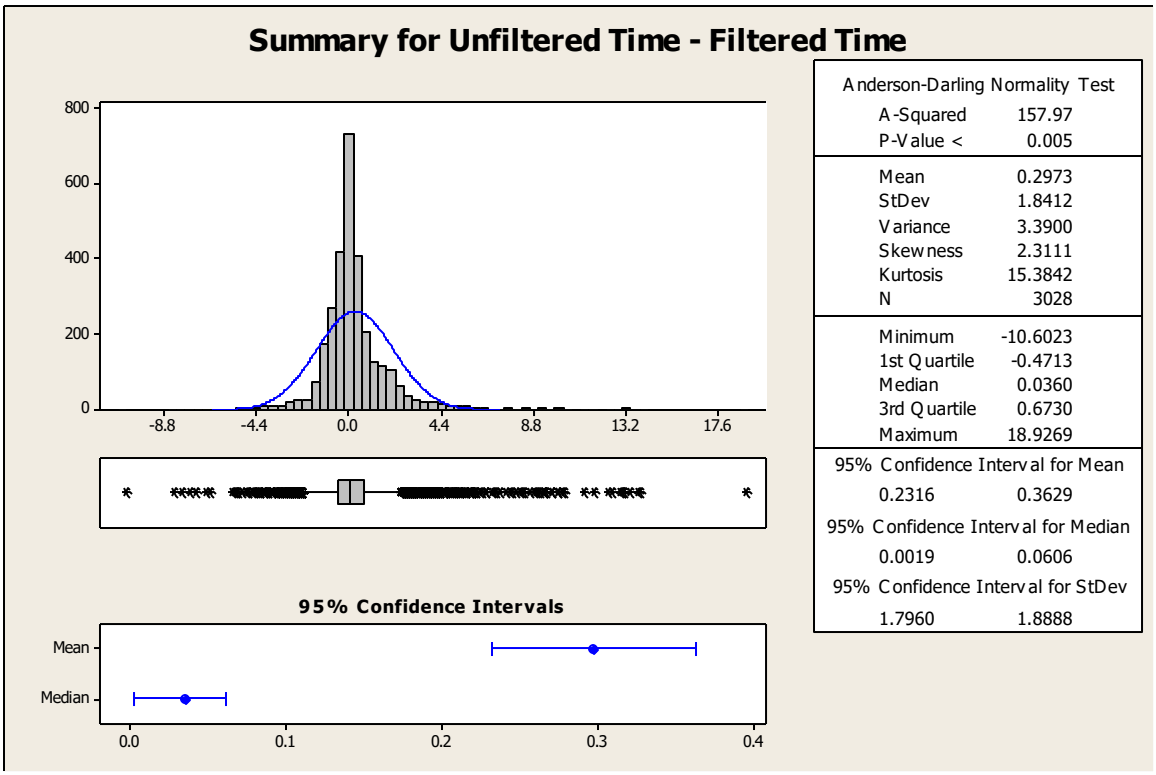

Figure 6: Test #2, Summary for Unfiltered Time - Filtered Time

From the histogram in the top left of this chart, it is unclear if there is a change page load time due to filtering. We can proceed with this test of hypothesis:

H_0: $u = u_0=0$ there is no change in mean webpage load time with filtering
H_1: $u \neq u_0=0$ there is a change in mean webpage load time with filtering

The paired z-Score was calculated with:

$$z = \frac{\bar{x}_r - u_0}{s_r / \sqrt{n}}$$

Where $\overline{x_r}$ is the sample mean of the paired differences of Filtered Time minus Unfiltered Time.

Where s_r is the sample standard deviation of the paired differences of Filtered Time minus Unfiltered Time.

Rejection region for these tests will be set at z > 3. This will provide us with over 99% confidence.

In this case z=8.9 and p-value < 0.001. There is sufficient evidence to reject the null hypothesis. The positive value for z indicates that there is still a savings in page load time due to filtering.

This indicates that with filtering, the page load time will decrease.

4.4. Conclusions for test #2

This indicates that there is a net increase in page size due to this method of filtering. This is counter-intuitive as we would expect a similar result as in Test #1. This will require further investigation.

The monitoring for this test occurred between the Linux client and the switch (see Figure 7). An increase in traffic at this point may be caused by extra traffic being sent by the Privoxy server, and not representative of the amount of traffic entering the network from the ISP. It is worth investigating the cause of the increase in traffic and whether or not it is actually being caused by the proxy's configuration.

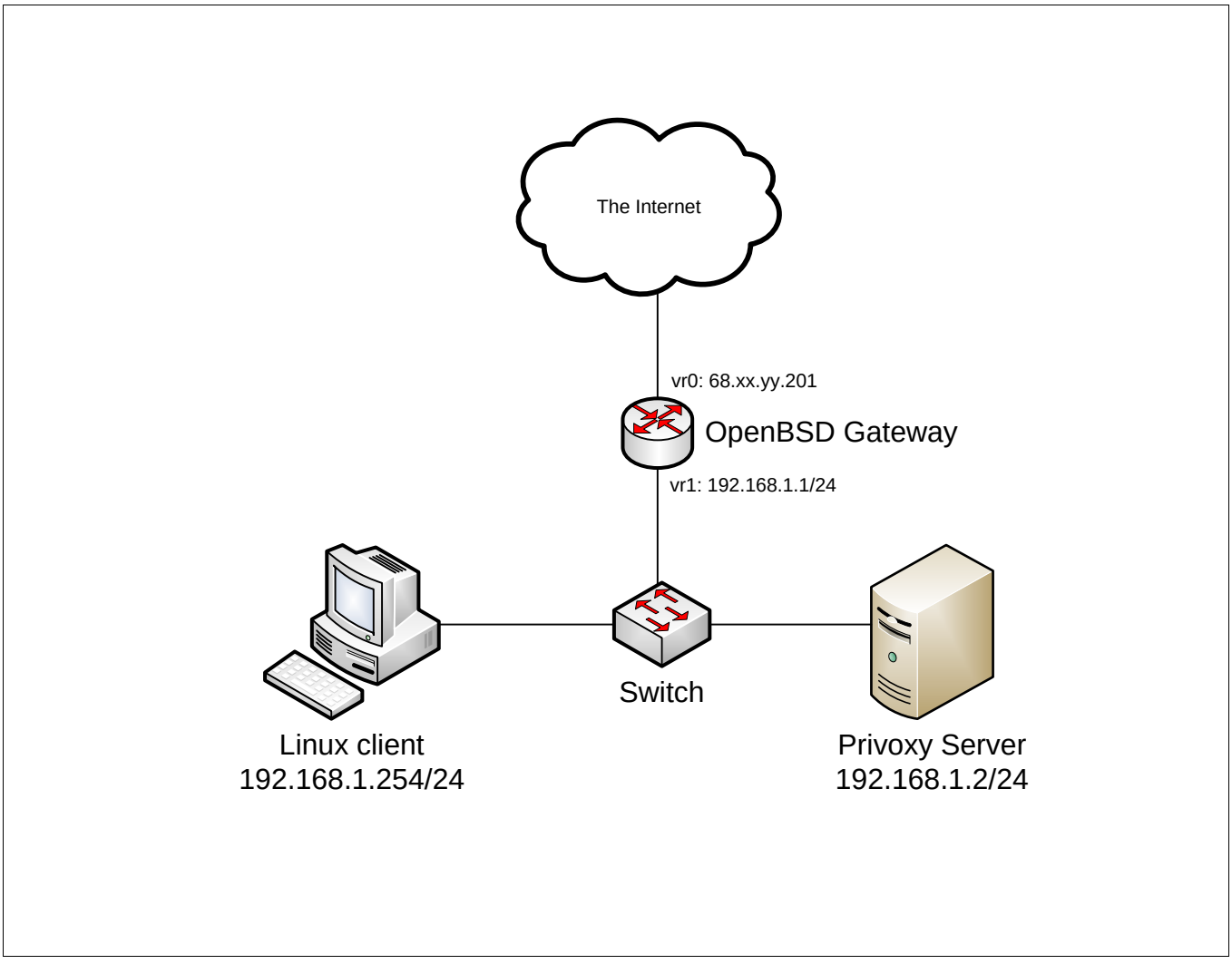

Figure 7: Network Diagram

5. Test #3 – Privoxy-blocklist with a Modified Configuration

5.1. Description

In this test, the goal was to first find likely problems with the Privoxy-blocklist configuration that could lead to increases in the page size. With the problems resolved, the goal would be to re-benchmark using the new configuration to see if it could rival or even approximate the results from the first test.

After some limited packet capture using WireShark[8], I noticed two things:

1) There was often a significant delay in the TCP stream when using Privoxy (Figure 9) over non-proxied traffic (Figure 8)
2) Extra data was being sent to the client web browser when data was being blocked. This behavior is reported in the Privoxy user community as discussed in ConSeanery, 2008[9].

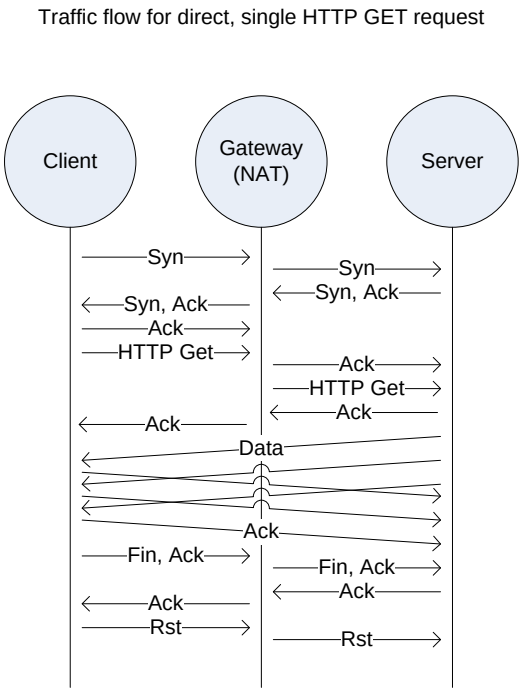

Figure 8: TCP Traffic Flow without proxy

ConSeannery provided several suggestions of how to adjust the configuration to reduce the amount of traffic being sent from Privoxy to the client workstation. Most of these involved adjusting parameters to suppress information from Privoxy that indicated that there was blocked traffic on a page. By default, Privoxy is configured to send warning messages to users when content is blocked. This is likely the cause of the additional bytes transferred when blocking advertising content.

[8] http://www.wireshark.org/ - WireShark allows users to do traffic analysis.
[9] http://www.fritscher.ch/blog/2008/09/03/google-chrome-adblock-with-privoxy/ - User: ConSeannery, 2008.

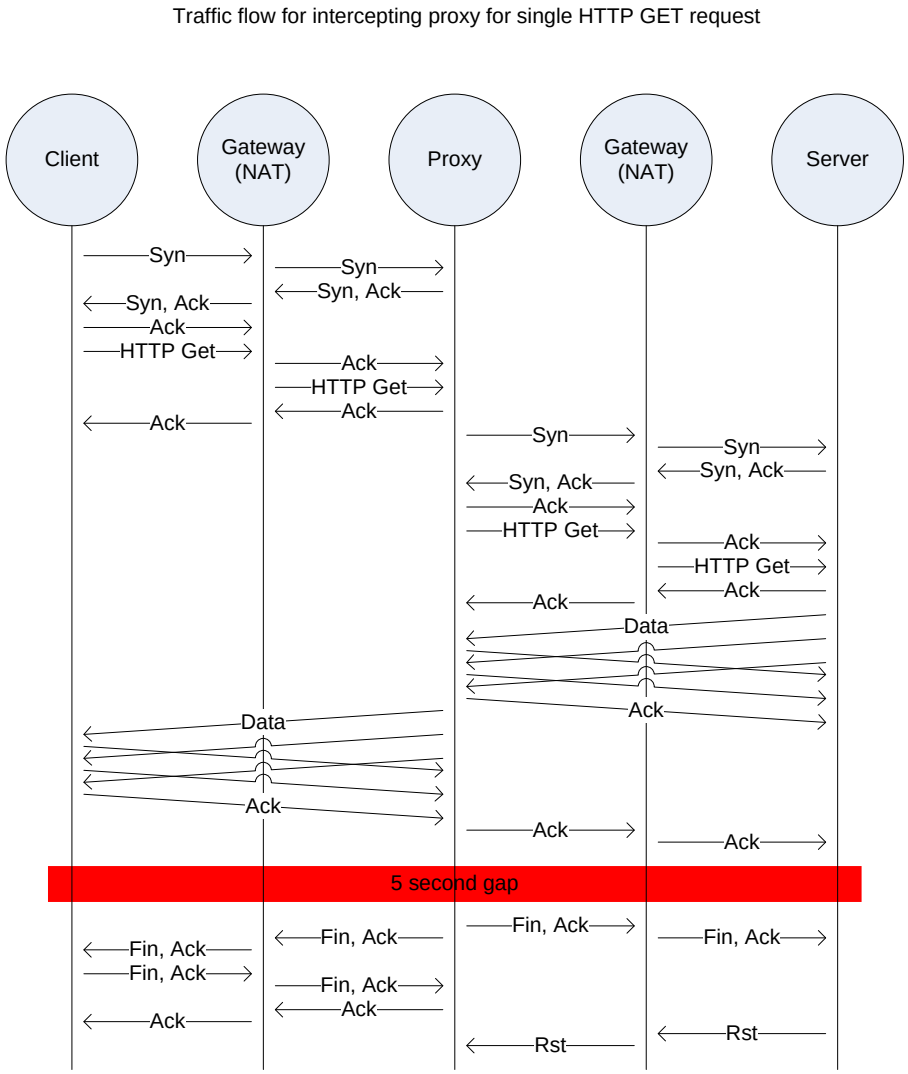

Figure 9: TCP Traffic flow with proxy

5.2. Methodology specific to test #3

I found the cause of the delay in the TCP stream to be a setting that is enabled by default. In the file "config" there is a field named "keep-alive-timeout" that is set to five seconds. This explained the significant delay in the TCP stream as shown in Figure 9. As this would not cause delays in processing time as was originally anticipated, it was left unchanged. This setting would leave the connection to the server open for five seconds to streamline any future connections to the server (skipping the TCP handshake that would otherwise be required), thus it is beneficial.

As per ConSeannery, in /etc/privoxy/default.action in the "standard.Cautious" section I modified:

```
+set-image-blocker{pattern}
```

to be:

```
+set-image-blocker{blank} \
```

and added inserted this line immediately below:

```
+handle-as-empty-document \
```

These changes prevented Privoxy from inserting additional header information when items were blocked. This should effectively decrease the number of bytes transferred.

By removing some of the default filters, I hoped to decrease processing time. In /etc/privoxy/match-all.action, I commented-out the following lines:

```
# +deanimate-gifs{last} \
# +filter{refresh-tags} \
# +filter{img-reorder} \
# +filter{banners-by-size} \
# +filter{webbugs} \
# +filter{jumping-windows} \
# +filter{ie-exploits} \
```

These lines are designed to do deeper inspection to the content than was necessary for ad blocking. The above filters modify content to remove some animations, analyze content based on heuristics which may not be beneficial and is likely not required. I find it unlikely, for example, that sites in the Alexa top 200 would publish known security vulnerabilities for some browsers (“ie-exploits”).

5.3. Results

The results in terms of bytes transferred:

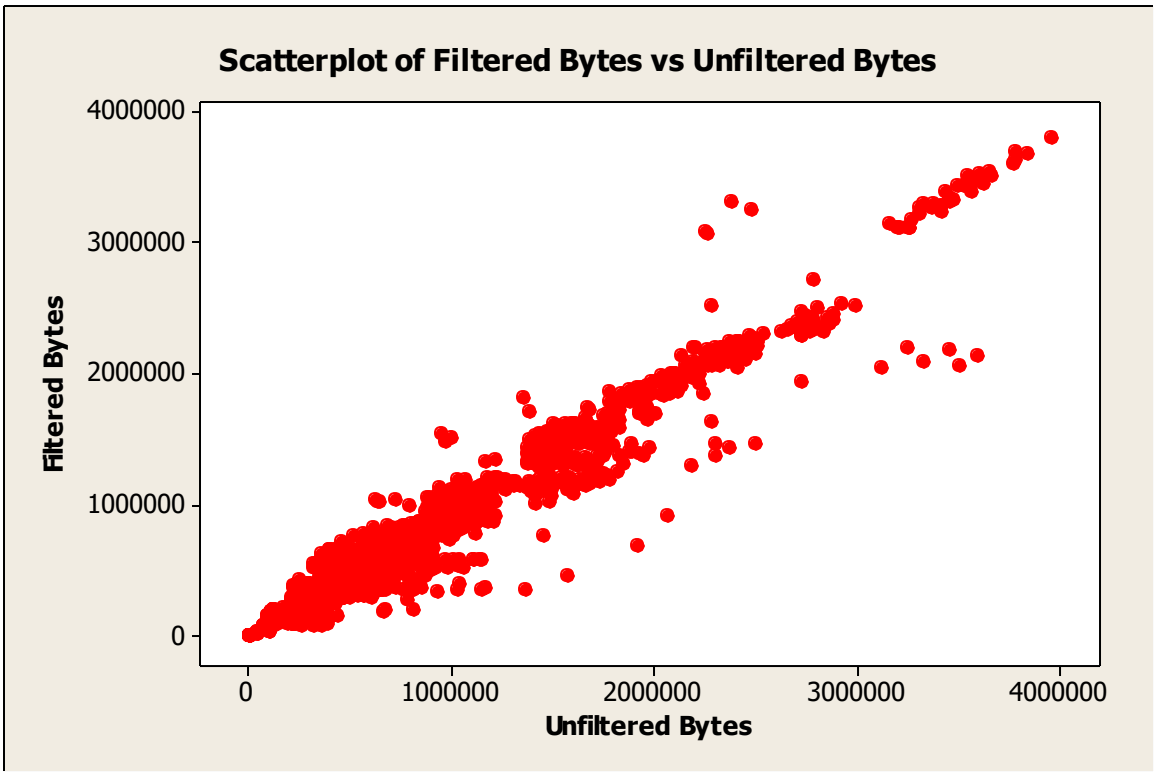

Figure 10: Test #3, Scatterplot of Filtered Bytes vs Unfiltered Bytes

Again, this appears to be a near-linear relationship. Examining the statistics for the difference of Filtered Bytes minus Unfiltered Bytes gives us:

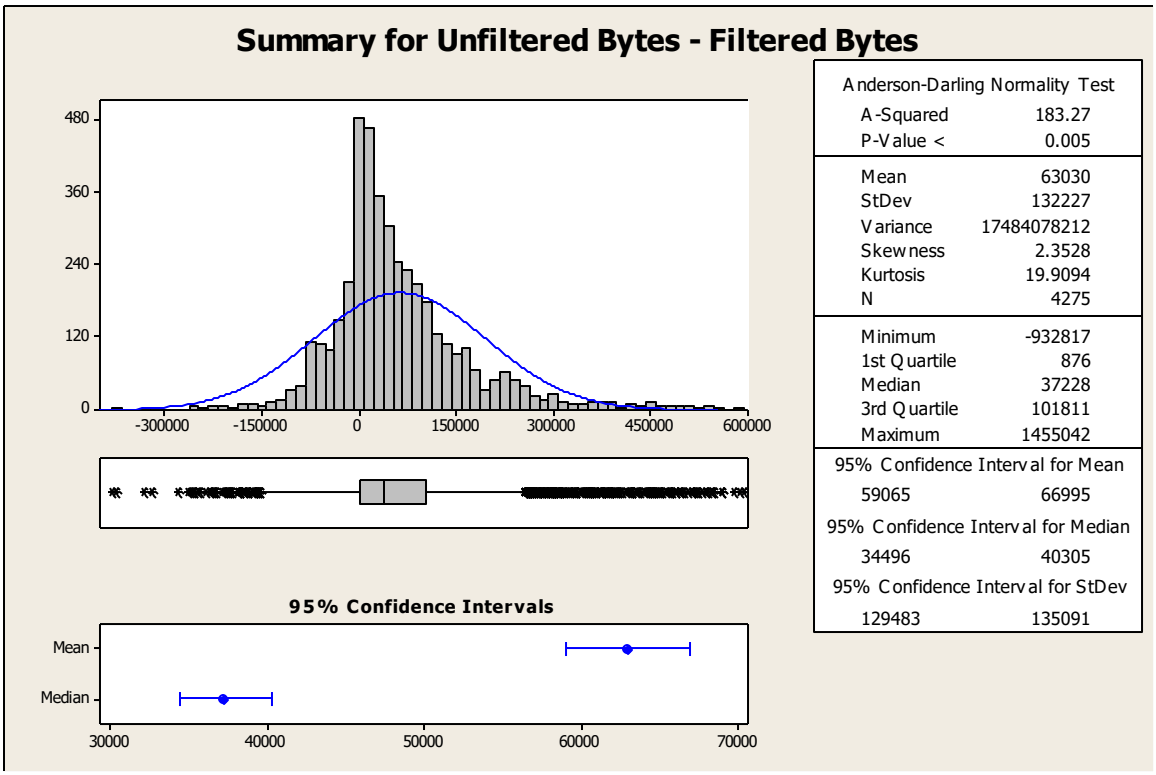

Figure 11: Test #3, Summary for Unfiltered Bytes - Filtered Bytes

As in the first test, the histogram in the top left shows how often there is a decrease in page size due to filtering (any value greater than zero) and how often there is an increase in page size due to filtering (any value below zero). Let's test the hypothesis that there is a decrease in page size:

H_0: $u = u_0 = 0$ there is no change in mean webpage size with filtering
H_1: $u > u_0 = 0$ there is a decrease in mean webpage size with filtering

The paired z-Score was calculated with:

$$z = \frac{\bar{x}_r - u_0}{s_r / \sqrt{n}}$$

Where $\overline{x_r}$ is the sample mean of the paired differences of Filtered Bytes minus Unfiltered Bytes.

Where s_r is the sample standard deviation of the paired differences of Filtered Bytes minus Unfiltered Bytes.

Rejection region for these tests will be set at $z > 3$. This will provide us with over 99% confidence.

In this case, $z=31.2$ and the p-value < 0.001. This is sufficient to reject the null hypothesis.

This indicates that this is a net decrease in page size due to filtering traffic with this method.

The next hypothesis that must be examined is whether or not there is a decrease in page load times due to filtering.

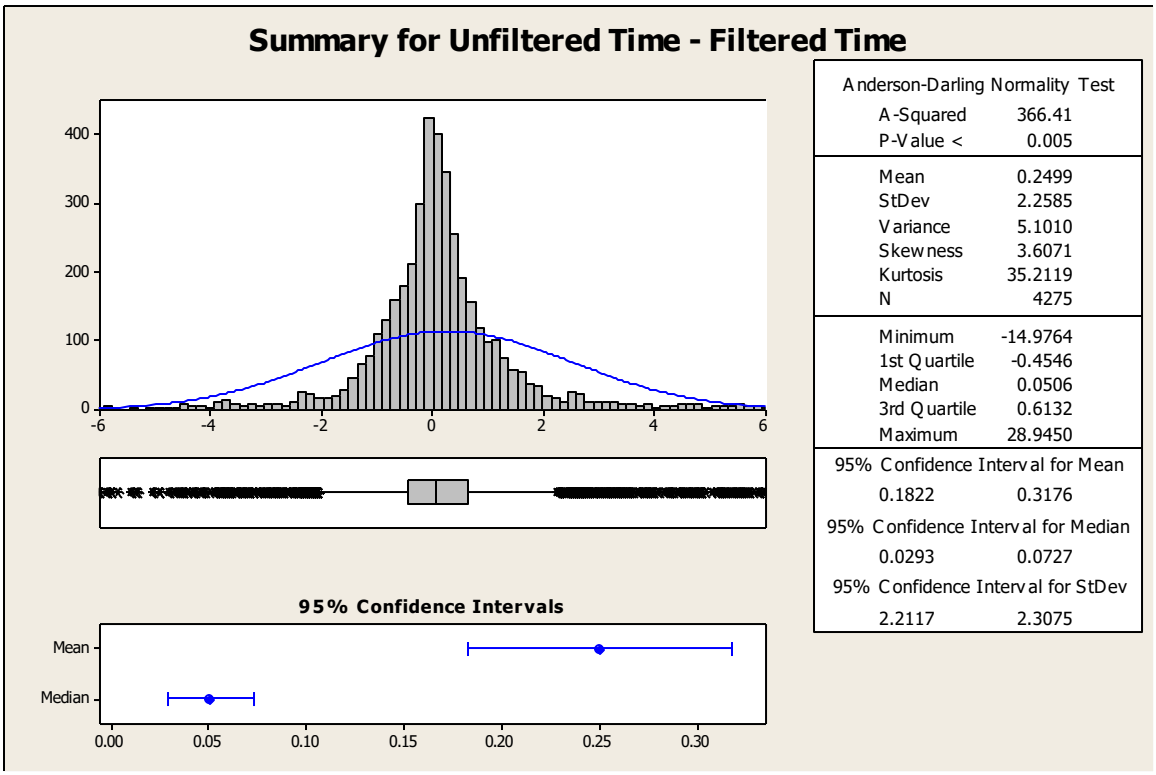

Figure 12: Test #3, Summary for Unfiltered Time - Filtered Time

The histogram in the top left is tending to the positive side, which suggests that there is a decrease in page load time. We can proceed with this test of hypothesis:

H_0: $u = u_0=0$ there is no change in mean webpage load time with filtering

H_1: $u > u_0=0$ there is a decrease in mean webpage load time with filtering

The paired z-Score was calculated with:

$$z = \frac{\bar{x}_r - u_0}{s_r / \sqrt{n}}$$

Where $\overline{x_r}$ is the sample mean of the paired differences of Filtered Time minus Unfiltered Time.

Where s_r is the sample standard deviation of the paired differences of Filtered Time minus Unfiltered Time.

Rejection region for these tests will be set at $z > 3$. This will provide us with over 99% confidence.

In this case $z=7.2$ and p-value < 0.001. There is sufficient evidence to reject the null hypothesis.

This indicates that with filtering, the page load time will decrease.

5.4. Conclusions for test #3

This type of filtering is successful in decreasing the amount of data transferred when loading webpages, and also shows a decrease in page load time.

6. Further Observations

A linear regression for bytes transferred of both Test #1 and Test #3 results (with a p-value of 0.0 for 95% confidence interval) shows:

- Adblock Plus: Filtered Bytes = 15065 + 0.878 Unfiltered Bytes
- Configured Privoxy-blocklist: Filtered Bytes = 4256 + 0.906 Unfiltered Bytes

This suggests that further improvements may be possible with Privoxy-blocklist in order to approach a more optimal result as with Adblock Plus. However, many users may be satisfied with the approximate 10% savings in bytes transferred when ads are blocked.

A linear regression for page load time of both Test #1 and Test #3 results (with a p-value of 0.0 for 95% confidence interval) shows:

- Adblock Plus: Filtered Time = 0.132 + 0.837 Unfiltered Time
- Configured Privoxy-blocklist: Filtered Time = 0.830 + 0.694 Unfiltered Time

Upon examining the order of the rules for the Privoxy configuration, I noticed that the rules generated by “Provoxy-blocklist” were in lexicographical order. This ordering may be sub-optimal for generic filtering. It may be possible to decrease the amount of time spent analyzing the rules (and thus decrease the page load time futher) by simply putting the most commonly encountered rules first in the list. I leave this as an exercise for the future.

7. Project Conclusions

Adblock Plus does not require network infrastructure to be implemented, in contrast, it must be installed and configured on each workstation. It can be configured for automatic updates to take advantage of any new ad-filters that the developers create. It is not available on all web browsers which may be considered a limitation. Use of Adblock Plus results in fewer bytes transferred, and a lower page load time.

Privoxy-blocklist requires network infrastructure in order to filter advertisements for multiple workstations. It can also be configured for automatic updates. Privoxy-blocklist can be used with any modern (HTTPv1.1 or higher) web browser. In its default state, there is no benefit in terms of bytes transferred for web pages when advertisements are blocked. With a modified configuration, benefits to both total page size and page load time are significant.

8. References

Alexa, The Web Information Company, www.alexa.com

Adblock Plus, www.adblockplus.org

IE7Pro, www.ie7pro.com

Privoxy, www.privoxy.org

Andrwe.org, www.andrwe.org

OpenBSD, www.openbsd.org

Wireshark, www.wireshark.org

Boris Fritscher's blog, www.fritscher.ch

Appendix A: /root/test.sh

```
#!/bin/bash

export PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
TESTID=test1-redo
#TESTID=test2

function clear-cache {
        find /root/.mozilla/ -type d | grep -i cache$ | xargs -n 1 rm -R
        find /root/.mozilla/ -type f | grep -i session | xargs -n 1 rm
        find /root/.mozilla/ -type f | grep -i cookies | xargs -n 1 rm
}

function turn-ads-off {
# make sure that ad filtering is turned on
        # test 1:
        cp /root/.mozilla/firefox/adblock-on/* /root/.mozilla/firefox/so1xpvu5.default/

        # test 2:
        #/usr/bin/ssh 192.168.1.1 'pfctl -t adfree -T add 192.168.1.254'
}

function turn-ads-on {
# make sure that ad filtering is turned off:
        # test 1:
        cp /root/.mozilla/firefox/adblock-off/* /root/.mozilla/firefox/so1xpvu5.default/

        # test 2:
        #/usr/bin/ssh 192.168.1.1 'pfctl -t adfree -T delete 192.168.1.254'
}

for SITE in `cat /root/top200-sorted.list`
do
        echo $SITE
        mytime=`date '+%s'`

        turn-ads-on
        clear-cache

        # Start the listener
        /root/webpage-bench.sh /root/$TESTID/"$SITE"_unfiltered $mytime &
        /root/firefox-wrapper.sh $SITE
        sleep 5

        turn-ads-off
        clear-cache

        # Start the listener
        /root/webpage-bench.sh /root/$TESTID/"$SITE"_filtered $mytime &
        /root/firefox-wrapper.sh $SITE
        sleep 5
done
```

Appendix B: /root/firefox-wrapper.sh

```
#!/bin/bash

SITE=$1

export XAUTHORITY=/home/tpb/.Xauthority
export DISPLAY=:0

/usr/bin/firefox $SITE
```

Appendix C: /root/webpage-bench.sh

```
#!/bin/bash

FILENAME=$1
ATTEMPTID=$2

tcpdump -lttni eth0 host 192.168.1.254 and tcp and port 80 | \
  sed -e "s/ .*, length//" | ./benchmark.py $FILENAME $ATTEMPTID
```

Appendix D: /root/benchmark.py

```
#!/usr/bin/python

import os, subprocess, signal, sys

MAXDELTA = 1.000

if len(sys.argv) != 3:
  print "Usage: <program> <file> <attempt-id>"
  exit()

try:
  outfile = open(sys.argv[1], "a")
except:
  print "ERROR opening file for append."
  exit()

killfirefox = subprocess

FIRSTLINE = ""
LASTLINE = ""
RECENTLINE =""
totalsize=0
firsttime=0
lasttime=0


def getline():
  try:
    line = raw_input()
    return line
  except:
    return "No data"

stop = 0

while stop==0:
  signal.signal(signal.SIGALRM, getline)
  signal.alarm(10)
  RECENTLINE = getline()
  if RECENTLINE == "No data":
    stop = 1
  else:
```

```
    if firsttime == 0:
      firsttime = float(RECENTLINE.split()[0])
      recentsize = int(RECENTLINE.split()[1])
      lasttime = float(RECENTLINE.split()[0])
      totalsize += recentsize

    else:
      recenttime = float(RECENTLINE.split()[0])
      recentsize = int(RECENTLINE.split()[1])
      if recenttime - lasttime < MAXDELTA :
        totalsize += recentsize
        lasttime = recenttime
      else:
        stop = 1

outfile.write("%d %d %0.4f\n" % ( int(sys.argv[2]), totalsize,
lasttime - firsttime))

os.system("/usr/bin/pkill tcpdump")
os.system("/usr/bin/pkill firefox")
```