Revisiting the Self Similar Nature of Web Traffic with Modern Browsers and Web pages

Sarah Tanveer
sarah.tanveer@wisc.edu

Nida Tanveer
ntanveer@wisc.edu

ABSTRACT

To be able to make important decisions about the design of the Internet, it is important to always understand the behavior of network traffic. While the self-similar nature of Web traffic is well-established and well-studied, it is still interesting to observe network traffic in light of changes in the landscape of the Internet. The complexity of web pages and the strategies employed by web browsers are different now. Our project will simulate users browsing popular websites, capture HTTPS traffic and analyze how factors like prefetching, and local caching impact the self-similarity of web traffic in the context of modern browsers and webpages. To summarize, our research question boils down to this: **Have strategies employed by modern browsers and web pages such as prefetching and local caching impacted the self-similar nature of web traffic?** In our results, we find a Hurst parameter of greater than 0.60 for all runs, meaning that for all our tests, even with prefetching and caching, traffic remains self-similar.

ACM Reference Format:
Sarah Tanveer and Nida Tanveer. 2025. Revisiting the Self Similar Nature of Web Traffic with Modern Browsers and Web pages. In . ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 OVERVIEW

The seminal paper "On the Self-Similar Nature of Ethernet Traffic" [7] by Leland, Taqqu, Willinger, and Wilson was published in 1994 and it fundamentally changed the understanding of network traffic patterns which, at the time, were believed to follow a Poisson distribution. They showed that Ethernet LAN traffic was statistically self-similar. Understanding the nature of Internet traffic is paramount when designing protocols for difficult problems like congestion control, conducting measurement studies, and exploring research topics in networking.

More than three decades later, the Internet landscape has changed drastically. Shifting from primarily a military and academic tool, it has evolved into a massive global infrastructure providing value to almost every aspect of our daily lives. The traffic on the Internet is incredibly diverse, from high-quality video streaming, cloud computing, IoT sensor data, social media consumption and much more. This high workload has led to optimizations such as local caching, prefetching, lazy-loading etc, to reduce latency and improve user experience. Most of the work done in the self-similarity space that we have mentioned in our Literature Review is from about two decades ago and since then there have been many changes to the complexities of web pages and browsers that we would like to explore. Our work is recreating the paper by Leland et al. [7] but for web traffic. We are also extending it by testing different browsers in order to capture the effects of prefetching and caching. We did not have access to real-world users to monitor web traffic, so we will simulate users to make requests to websites and monitor traffic to collect our data. We assume that this method would be suitable as our user equivalents would still be making requests to real websites. We expect that if our simulation emulates users well enough, we will see self-similar traffic in the default case. After we have established this, we want to explore the impact of caching and pre-fetching as well as the impact of the diurnal pattern of network traffic. We decide to primarily explore the impact of prefetching and caching. We do this by running our experiment with prefetching enabled vs disabled using options provided by Selenium. For caching, we rerun a previous experiment while using a user's saved browser cache. In our results, we find a Hurst parameter of greater than 0.60 for all runs, meaning that for all our tests, even with prefetching and caching, traffic remains self-similar.

2 RELATED WORKS

In this section, we explain key terms required to understand our paper. We also situate our work in relevant literature and provide the necessary background needed to do so. We

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

identify key trends in the literature and highlight gaps where our research may add necessary insight.

On the Self-Similar Nature of Ethernet Traffic [7]: The data set used consists of LAN traffic data consisting of hundreds of millions of Ethernet packets which were accurately timestamped (up to 100μs and 20μ). This data was collected on several Ethernet LAN's at the Bellcore Morristown Research and Engineering Center between August 1989 and February 1992. First, they describe their dataset in detail, making sure to comment on the impact of new changes such as the switch to "dataless" workstations - which increases network traffic. Before the publication of this paper in 1994, it was believed that Ethernet traffic, similar to circuit-switch traffic, followed a Poisson-like distribution i.e., it smoothed out on large enough time scales. Through statistical analysis, they show that Ethernet traffic is statistically self-similar, meaning that aggregating streams of this data across different time scales intensifies it's burstiness instead of smoothing it out. Then in the following section, they state the mathematical definition of self-similarity, identify classes of stochastic models that are capable of accurately describing the self-similar behavior of the traffic measurements, and illustrate statistical methods for analyzing self-similar data sets.

In general, the authors describe a self-similar process as having the following properties:

(1) the variance sample mean decreases more slowly than expected
(2) it exhibits long-range dependence, i.e. bursts from the past still impact future bursts
(3) burstiness exists at all time scales

These properties are measured by Hurst's parameter, where an H value of greater than 0.5 shows self-similarity. The authors use statistical analysis on their dataset which include graphical methods such as variance-time plot, pox plots, periodgrams as well as maximum likelihood estimation and Whittle's Method, to establish long-range dependence and show that burstiness exists in Ethernet traffic in all time scales. They find that graphical methods are comparably accurate. Through the contributions in this work, the community gained valuable insights into the true nature of Web traffic which is essential to know for traffic modeling, congestion control and many other research problems in the space of networking. Since they found similar results for the TCP traffic in their dataset, we have decided to use HTTPS traffic for our work. We will use similar statistical techniques on HTTPS Web traffic to analyse how the self-similarity of Internet traffic has been impacted by the use of prefetching, lazyloading and local caching by modern browsers and webpages.

Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes: This paper proposes that the self-similar nature of web traffic is due to file sizes, the effects of local caching, and user behavior. The authors postulate that the set of available files is the most important factor in determining whether the distribution of traffic is heavy-tailed and hence is self-similar [5]. Local caching is a mechanism by which certain web objects can be stored at the end host so that the next time the user requests the object it could be fetched directly instead of having to make a network request (based on the "freshness" of the object). Caching is one the main factors that determine the size of available files and in our work, we plan to analyse the impact of local caching on self-similarity as it has an impact on the number of network requests made by the user. The paper naturally only tests the caching scheme employed by the Mosiac browser. Today's browsers employ different caching schemes that can be tested [6].

The Network Effects of Prefetching: This paper evaluates how different prefetching schemes can impact the performance of the network. Naive prefetching (getting all the files requested in a session) can increase queuing effects thereby increasing network delay. For our work, we are interested in how it explores the fact that prefetching extends both ON and OFF periods, increasing burstiness, thereby contributing even more to self-similarity. However, it was performed on a simulation and it also commented on how "rate-controlled prefetching" could perhaps smooth out traffic. This may be because prefetching can "predict" the OFF periods, reducing user think times. We know these factors impact self-similarity and prefetching is one of the factors we plan on evaluating in our work. Modern prefetching techniques may be more efficient which is why we believe it may be worth looking into and how web traffic can be potentially smoothed out.

Generating Representative Web Workloads for Network and Server Performance Evaluation
This paper presents SURGE, a tool for emulating real-world traffic from users. It takes into account crucial aspects like file size distribution, request size distribution, file popularity, embedded references, caching, idle period, concurrent connections, and produces self-similar traffic. For our work, we are interested in the insights that the paper provides about user behavior that we plan on incorporating into our data collection pipeline. For any measurement study that aims to analyze the nature of web traffic, it is crucial to collect a representative data set. One way this can be done is by using a trace-based approach (pre-recorded records of past workloads) or an analytic approach which involves using mathematical models for various workload characteristics

and generating outputs that adhere to those models [4]. Although [4] outlines the design of a workload generator, the key insights are still applicable to our work since they highlight key characteristics we should consider while simulating web usage. [4] emphasizes two main aspects of the workload: User Equivalents and Distributional Models. A single user is defined as a single process in an endless loop that alternates between making requests for Web Files and lying idle.

Each UE also needs a set of probability models. For example, the sizes of the files is an important consideration since they should follow a heavy-tailed distribution if we want to accurately represent file sizes on the Web. The duration of idle times must be accurately modeled. There can be two types of idle times: inactive and active. We refer to "idle" periods and as OFF time. Inactive OFF time is the time between requests of Web objects (this can be thought of as natural user "think" time). Active OFF time refers to the time between a request was made and the user is waiting for a response.

For the scope of our project, we will not be generating workloads directly but rather simulating user behavior. Therefore the discussion in [4] is helpful since it highlights some key characteristics we cannot ignore. For our project, the most relevant ones would be simulating inactive OFF times (Pareto distribution) and popularity of the files being requested.

Performance analysis of a client-side caching / prefetching system for Web traffic. This paper investigates the impact of client-side caching and prefetching on web traffic performance [1] . They focus on the trade-off between response time and network congestion because although prefetching reduces the response time of a requested document, it also increases the network load, as some documents will be unnecessarily prefetched. They show that prefetching all documents with access probabilities greater than a given threshold value does not always lead to the minimum access delay. They also show that the access delay improves with increasing number of prefetched documents until a certain point, after which the trend reverses. This shows that there is an optimal number of optimal number of prefetched documents exists, which minimizes response time while controlling excess traffic. This study is relevant to our research as it highlights how adaptive prefetching affects web traffic which gives us a baseline to investigate whether self-similarity persists in web traffic even today.

Review and Analysis of Web Prefetching This work presents a summary of the documentation Firefox, Chrome and Internet Explorer and how they claim to carry out prefetching. The authors then conduct a small-scale experiment to confirm whether these browsers actually prefetch objects. They find that Firefox was the only browser that always prefetches the expected objects in their tests, despite Chrome's default setting having prefetching enabled. Thus, we used Firefox to test traffic with prefetching enabled vs disabled.

3 METHODOLOGY

3.1 Simulation setup

For our approach, we simulate users (user equivalents) by using Selenium and three different browsers (Chrome, Edge and Firefox). We write scripts to browse through web pages and social media sites. We chose three broad categories of websites i.e. social media/short-form content, search engines, and news sites. We decided on these categories by examining the top 30 or so websites/URLs from popular website datasets like Tranco, CrUx and Alexa Top 100 and most websites fell into these categories. We chose 3 different browsers for our experiments since we wanted to investigate if the browser type had any notable effects on the nature of the web traffic, prefetching and local caching. Different browsers employ these strategies in different ways as explored in [6] and shown in their documentation [3, 8].

Algorithm 1 Simulate Browsing Behavior

```
Open website
for i = 1 to Random(Number of Actions) do
    Randomly choose between Scroll or Wait
    if Scroll chosen then
        Choose direction (up or down) and scroll for random length
        Wait for a random duration (Inactive Wait)
    else
        Wait for a random duration (Inactive Wait)
    end if
end for
Choose a random link on the page and open it
for i = 1 to Random(Number of Actions) do
    Randomly choose between Scroll or Wait
    if Scroll chosen then
        Choose direction (up or down) and scroll for random length
        Wait for a random duration (Inactive Wait)
    else
        Wait for a random duration (Inactive Wait)
    end if
end for
```

For the website automation, we utilised the On/Off Model described in [2]. The general structure of the experiments is shown in Algorithm 1. The inactive wait times were sampled from the Pareto distribution.

3.1.1 *Websites.*

(1) **Tiktok:** This was the choice for the social media/short-form content category. Its experiment was different from the other websites due to the website structure. There was no random scrolling up or down, just scrolling down to go to the next video or watching the video for a random duration. The watch time was sampled randomly. For Tiktok, there was minimal caching during the second run because every the same user was rarely served the same videos.
(2) **The Guardian:** The Guardian was our pick for the news category. its methodology was very similar to the algorithm described in Algorithm 1. To reduce the number of articles our users visited, we picked only 5 categories from the homepage for the users to pick articles from. For the caching run, we simply saved these links for the user to visit again.
(3) **Google:** Our choice for the search engine category, again, had a very similar structure as the algorithm described in Algorithm 1, except it had the added action of "Search". We picked 10 additional popular websites from the popular websites datasets randomly for a user to search for and then randomly select a link from.

3.2 Traffic capture methods and tools

For capturing traffic, we make use of tcpdump which is a powerful tool used for capturing network traffic [9]. We listen on the ports 80, 8080, and 443 since we wanted to only focus on HTTP/HTTPS traffic. We start running tcpdump after we start our browsing scripts. While browsing, we only have the scripts running and nothing else on the machine. This ensures that other traffic does not interfere with what we want to analyze. We capture traces of 30 minutes. Since the file sizes of the packet capture files (.pcap) is significantly large, loading these into memory for further analysis is cumbersome. Instead we choose to perform slicing - we only keep the first 128 bytes of the packet (i.e. the packet header). We are not interested in the contents of the payload for the first part of our analysis so this makes the files more manageable.

3.3 Experimental conditions

(1) **Browser:** We ran our simulation on three different browsers, namely Chrome, Firefox and Microsoft Edge. Modern browsers implement their own web optimizations, making this a natural way to compare the impact of different prefetching and caching techniques. However, this can be a bit of a black-box as we would not be sure as to what exactly causes the difference between any two browsers' traffic if we observed it. We can only make limited comments about factors that could lead to differences. For example, Chrome has pre-rendering turned on automatically, while Firefox doesn't. Microsoft Edge is based on Chromium so we expect it to be similar to Chrome.
(2) **Caching:** We run the experiment once and we log user actions. We save the cache of that run in "User Profiles", a tool provided by Selenium. We then re-run the experiment using the same user profile to re-use the cache and visit similar links (although not necessarily in the same order). This was not possible with TikTok as the For You page does not provide links to the videos you are watching, so it was not possible to manually replay the same videos in the second run.
(3) **Prefetching:** For prefetching, we understand from [6] that other browsers besides Firefox may not be implementing prefetching completely. So, we picked Firefox to be our control for prefetching. In the default settings, prefetching is enabled (although prerendering is not), and then we run the experiment again while manually disabling all forms of prefetching.
(4) **Time of Day:** We repeated the experiments once in the afternoon and once in the evening in an attempt to capture the impact of the diurnal cycle - a natural phenomenon that arises due to the schedule of the work day.

As a result of our experiments, we got pcap files of packets with timestamps. Our code can be found here: https://github.com/nidatx/comp-sci-740

3.4 Analytical methods

In the original paper by LTWW94, perhaps the most insightful plots were those shown in Figure 1 which the authors refer to as "pictorial-proof" of the self-similarity. This figure showed a sequence of simple plots of the packet counts aggregated over different timescales. We aim to recreate these plots.

Although these simple plots offer valuable insights, we must also mathematically quantify the self-similarity of the traffic. Before diving into the results of our experiments, it would be beneficial to provide a brief overview. The Hurst parameter (a.k.a Hurst exponent) is used as a measure of long-term memory of time series. It is a number from 0 to 1. $H = 0.5$ indicates a random walk, where past values have no predictive power for future values. A value of $H < 0.5$ indicated that there are no long-range dependencies. A value of $H > 0.5$ indicates persistence/long-range dependencies in the time-series. If we get a value of $H > 0.5$, it tells us that our time-series data is self-similar.

Mathematically, self-similarity manifests itself in a number of equivalent ways: (i) the variance of the sample mean decreases more slowly than the reciprocal of the sample size (slowly decaying variances), (ii) the autocorrelations decay hyperbolically rather than exponentially fast, implying a

non-summable autocorrelation function and (iii) the spectral density obeys a power-law behavior near the origin.

R/S analysis. They state that the graphical R/S approach is most useful and attractive due to its relative robustness against changes of the marginal distribution. This feature allows for practically separate investigations of the self-similarity property of a given data set and of its distributional characteristics [7]. The R/S analysis involves splitting the time series into non-overlapping blocks of size m, then calculating variation of each point from the mean and keeping a cumulative sum of these variations. We find the range by subtracting the maximum and minimum values of the cumulative sums. We divide this by the standard deviation to get the R/S statistic. We find the R/S statistic for each block and average these to get an average R/S value that corresponds to a certain m. They present a graph of $log(R/S)$ versus $log(m)$. An estimate $\hat{H}$ is given by calculating the gradient of the line of best-fit.

Variance-time analysis. They also present a variance-time analysis. Similar to the previous approach, this involves splitting the time-series data into non-overlapping blocks of a size m, calculating the mean of the block and finding the variance of this aggregated series. This is repeated for different values of the block size m. They plot $log(X^{(m)})$ against $log(m)$. The authors state that for second-order self-similar processes, the variances of the aggregated processes $X^{(m)}, m > 1$ decrease linearly (for large m) in $log - log$ plots against m with slopes arbitrarily flatter than -1. Values of the estimate $\hat{\beta}$ of the asymptotic slope between -1 and O suggest self-similarity, and an estimate for $\hat{H} = 1 - \frac{\hat{\beta}}{2}$.

We use both the R/S and variance-time analysis to estimate the Hurst parameter for our project.

4 EXPERIMENTS/DATA

The data we collected was through running for simulation with different configurations for 30 minutes each, with three concurrent users as well as a single user. It was difficult to scale the experiments because of the unpredictable nature of Selenium, and the limited number of devices we had (we could not use our device for anything else while the experiments ran). In total, we ran about 23 experiments (excluding some initial ones to test our scripts), but we will report on only a few. We conducted the same experiment with different configurations to test how multiple factors may impact self-similarity.

5 RESULTS

We conducted our analysis using the methods described in [7]. This includes recreating Figure 1 (the pictorial proof of self-similarity). We also quantify self-similarity using the Hurst parameter using the R/S as well as variance-time analysis. We calculate H using R/S analysis as well as variance-time plots. These are summarized in Table 1 and 2.

	Afternoon		Evening	
	First	Cache	First	Cache
Chrome	0.784	0.776	0.785	0.786
Edge	0.830	0.799	0.824	0.811
Firefox	0.793	0.791	0.839	0.843

Table 1: Hurst estimates from R/S analysis for First Run vs. Cached (3 concurrent users)

	Afternoon		Evening	
	First	Cache	First	Cache
Chrome	0.643	0.587	0.644	0.652
Edge	0.733	0.662	0.704	0.731
Firefox	0.675	0.666	0.773	0.656

Table 2: Hurst estimated from variance-time analysis for First Run vs. Cached (3 concurrent users)

	Non-Prefetched	Prefetched
R/S	0.804	0.793
Variance-time	0.733	0.675

Table 3: Hurst estimates for Firefox: Prefetched vs Not Prefetched (3 concurrent users)

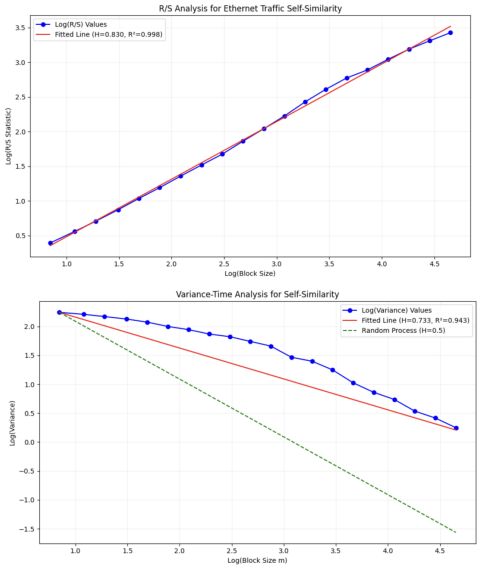

Figure 1: Graphical Methods for checking the self-similarity property - shown for an afternoon Edge run

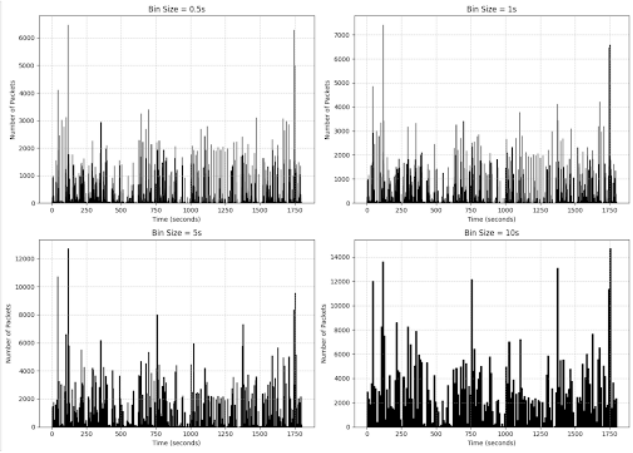

Figure 2: Pictorial Proof of self-similarity: packets per unit time for 4 different time scales (Using Chrome on default settings)

5.1 Pictorial Proof of Self Similarity

Figure 2 shows the pictorial proof of self-similarity i.e., packet traces remain bursty in nature regardless of the time-scale that they are aggregated at. We aggregated the packet arrival events on a granularity of 0.5s, 1s, 5s and 10s. As stated in the paper, this method provides a very simple method to visualize the self-similar nature of the data. We can see that the burstiness persists across different time-scales making each of the graphs show very similar patterns.

This packet trace was collected using Chrome on default settings on a first visit (so no caching). From what we can gather from documentation, by default Chrome uses prefetching and prerendering.

5.2 Analyzing Factors

The $\hat{H}$ values obtained from our R/S and variance-time analysis are summarized in Table 1 and 2 respectively. We notice that the estimates we obtained for the Hurst parameter using the variance-time analysis were lower than those obtained with the R/S analysis. This could be because the R/S analysis is more sensitive to short-term correlations and can be influenced by local trends. Variance-time analysis focuses more on how variability scales with aggregation level. Both these methods are also meant for infinite data so the estimates produced for a finite set of data might yield different estimates. This could also be due to the duration of our experiments (only 30 minutes) as compared to the original study that used traces that went up to 48 hours. However, this is not a point of concern since both methods agree on the same direction ($\hat{H} > 0.5$) which indicates self-similarity.

Table 3 shows $\hat{H}$ when we had prefetching enabled and disabled on the Firefox browser.

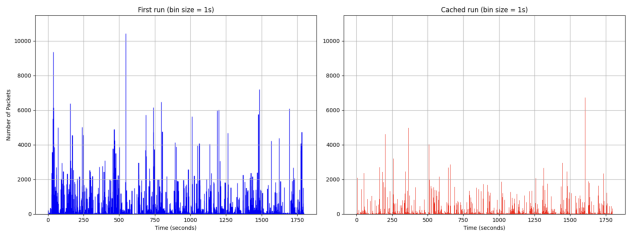

Figure 3: Packet frequencies for initial versus cached experiment

In this section, we also provide relevant visualizations. For the lack of space, we provide examples from a specific run which we mention in the text.

(1) **Diurnal Pattern:** We see that running our experiment during the afternoon versus the evening does not have any notable effect on $\hat{H}$ while using Edge and Chrome. For the R/S values of $\hat{H}$, the percentage difference for both is $< 1\%$ and for the variance time values of $\hat{H}$, the percentage difference for both is $< 5\%$. However for Firefox the percentage difference is 5.8% (R/S) and 14.5% (variance-time) between the afternoon and evening packet traces. Although we cannot be certain what might have caused this, it seems that Firefox's traffic patterns might be more sensitive to network conditions, which vary throughout the day due to diurnal internet usage patterns.

(2) **Caching:** From Table 1 and 2 we see that for Edge and Firefox, the $\hat{H}$ values for the cached runs are always less than those on the first visits. However, this is not true for Chrome - we see that $\hat{H}$ for the cached evening runs is 0.12% (R/S) and 1.2% (variance-time) greater than $\hat{H}$ on the first visit. Even though this is a very small percentage difference, it was quite unexpected as Edge is based on Chromium so we expected their results to be very similar. Figure 3 shows the packet arrival events aggregated over a time-scale of 1 second for an initial versus cached run (shown for an afternoon Edge run). Although the burstiness of the traffic persists across both scenarios , we see that there are significantly fewer packet arrival events in the cached run. This can be observed by the shorter and slightly more dispersed bars (which appear "thinner") in the cached run as compared to the initial run.

(3) **Prefetching:** We only conducted the prefetching experiments on Firefox because that was the browser that was identified in [6] to have actually been doing some prefetching. We disabled all forms of prefetching by following the documentation provided by Firefox using Selenium. We used a separate user for the disabled prefetch run so as to not use caching. We see that the $\hat{H}$ was very similar to the non-prefetch version. This

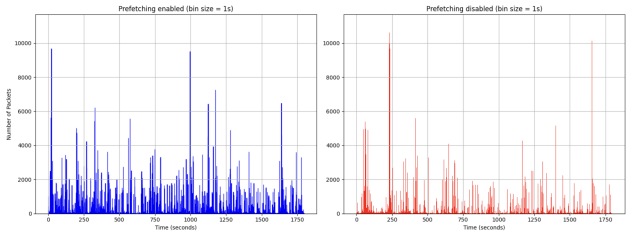

Figure 4: Packet frequencies for prefetching enabled vs. disabled

could also be because of the websites we chose. Perhaps they do not choose to prefetch too many objects. Although, we were under the impression that Google does prefetch some top links when a user searches for something. It is challenging to confirm whether an object was prefetched or not. Figure 4 shows the distribution of traffic when we have prefetching enabled and disabled. We see that the burstiness persists and both packet traces yield $\hat{H} > 0.5$ as shown in Table 3. However, we can also visually see that there the bars appear 'thinner' because the packet arrivals are more dispersed, with greater spacing between intense burst periods.

5.3 Comparison with Original Findings

Leland et al. reported Hurst parameters that typically range between 0.7 and 0.9 for their Ethernet traffic traces[7]. Our mean estimate is approximately 0.74 which falls within this range, suggesting that despite three decades of technological evolution, the self-similar nature of network traffic remains intact. This stability is remarkable considering the dramatic changes in network speeds (from Mbps to Gbps), content complexity (from simple HTML to dynamic multimedia pages) as well as client-side optimization techniques (like caching and prefetching).

In the paper, the authors propose that the underlying causes of self-similarity are due to the heavy-tailed distribution of file sizes and the ON/OFF patterns of user behavior. The fact that self-similarity remains suggests that these main causes remain the determinants of traffic patterns as a whole, regardless of technological advances.

6 CONCLUSION AND FUTURE WORK

In the future, we would like to extend our work by increasing the number of users in our simulation as well as adding more websites to make our data more representative. It would also be interesting to conduct similar experiments on mobile devices. The last step to fully establish self-similarity would be to conduct a similar large scale measurement study on real-world users as done by Leland et al [7]. To summarize, we conduct an emulation study to test the impact of modern browsers and webpages and strategies such as prefetching and local caching on the self-similarity of Web traffic. We wrote a script that simulates a user browsing a webpage with a number of user actions such as waiting, scrolling, searching, etc. After capturing network traffic data using tcpdump, we conduct R/S analysis and variance-time analysis to estimate the Hurst parameter to establish whether the traffic was self-similar. In our results, we find a Hurst parameter of greater than 0.60 for all runs, meaning that for all our tests, even with prefetching and caching, traffic remains self-similar. Our results indicate that cached runs generally exhibited lower Hurst parameters than non-cached runs. Our experiments with prefetching enabled and disabled yield similar values of $\hat{H}$. Our results indicate that web traffic today maintains it's self-similar nature regardless of technological advancements.

REFERENCES

[1] Balamash, A., Krunz, M., and Nain, P. Performance analysis of a client-side caching/prefetching system for web traffic. *Computer Networks 51*, 13 (2007), 3673–3692.

[2] Barford, P., and Crovella, M. Generating representative web workloads for network and server performance evaluation. SIGMETRICS '98/PERFORMANCE '98, Association for Computing Machinery, p. 151–160.

[3] Chromium Project. Chromium Design Documents: Very Simple Backend, 2024. [Accessed: 30-Apr-2025].

[4] Crovella, M., and Barford, P. The network effects of prefetching. In *Proceedings. IEEE INFOCOM '98, the Conference on Computer Communications. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Gateway to the 21st Century (Cat. No.98* (1998), vol. 3, pp. 1232–1239 vol.3.

[5] Crovella, M., and Bestavros, A. Self-similarity in world wide web traffic: evidence and possible causes. *IEEE/ACM Transactions on Networking 5*, 6 (1997), 835–846.

[6] Deng, Y., and Manoharan, S. Review and analysis of web prefetching. In *2015 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)* (2015), pp. 40–45.

[7] Leland, W. E., Taqqu, M. S., Willinger, W., and Wilson, D. V. On the self-similar nature of ethernet traffic. SIGCOMM '93, Association for Computing Machinery, p. 183–193.

[8] Mozilla. Firefox Networking: cache2 Documentation, 2024. [Accessed: 30-Apr-2025].

[9] The Tcpdump Group. Tcpdump and Libpcap. https://www.tcpdump.org/. Accessed: Apr. 20, 2025.