BrowsEm: Model-based Web Site Loading Emulation

Kilian Holzinger
holzinger@net.in.tum.de
Technical University of Munich
Germany

Florian Klein
kfl@net.in.tum.de
Technical University of Munich
Germany

Daniel Petri
petriroc@net.in.tum.de
Technical University of Munich
Germany

Stefan Lachnit
lachnit@net.in.tum.de
Technical University of Munich
Germany

Sebastian Gallenmüller
gallenmu@net.in.tum.de
Technical University of Munich
Germany

Georg Carle
carle@net.in.tum.de
Technical University of Munich
Germany

Abstract

Continuous research and engineering efforts aim to improve the performance of web networking protocols, such as TCP, QUIC, TLS, and HTTP. Performance measurements are conducted to assess the impact of changes to these protocols, their implementations, or the underlying network infrastructure. However, this is a challenging task due to the increasing complexity of web site deployments and browser and server software. In this work, we introduce BrowsEm, a web site loading emulator capable of reproducing page loading workloads. Its underlying model is based on data scraped from real web sites and takes network path characteristics, protocols, dependencies between HTTP transactions, and their individual timing aspects into account. Modularly structured and using wide-spread libraries like libcurl, it allows for testing a wide range of emulation parameters such as network conditions or protocol implementations and obtaining reproducible results. Our evaluation shows that the emulation introduces a relative error smaller than ± 0.25 for 80% of observed page load times. In a measurement campaign, we find that the model shows suitable robustness for artificially changed parameters.

CCS Concepts

• **Networks** → *Network performance modeling*; **Network experimentation**; **Network performance analysis**; **Network measurement**; **Application layer protocols**; **Transport protocols**.

Keywords

web, browsers, emulation, measurement, protocols

ANRW '25, Madrid, Spain

ACM ISBN 979-8-4007-2009-3/2025/07
https://doi.org/10.1145/3744200.3744759

ACM Reference Format:
Kilian Holzinger, Florian Klein, Daniel Petri, Stefan Lachnit, Sebastian Gallenmüller, and Georg Carle. 2025. BrowsEm: Model-based Web Site Loading Emulation. In *Applied Networking Research Workshop (ANRW '25), July 22, 2025, Madrid, Spain.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3744200.3744759

1 Introduction

Web performance is important for the user experience and the success of web applications. The demand for improvements has led to the development of new protocols such as QUIC and HTTP/3 or extensions to the web stack such as transaction multiplexing, multipath transport protocols, prioritization schemes, or post-quantum cryptography. Engineering and research efforts continuously assess the impact of new developments, identify limitations, and provide proof-of-concept implementations. However, accurately evaluating the influence of changes to the network stack on web applications remains challenging. Modern browsers are complex software systems, making it hard to add functionality as proof of concepts to network subsystems such as the transport or HTTP protocol implementation. Additionally, reproducibility of in-browser measurements is a challenging endeavour. In contrast to that, fully synthetic evaluation approaches fail to reflect realistic workload patterns.

There is a methodological gap to enable experimentation with the web network stack while still being able to impose realistic and reproducible application workloads. An approach is needed to emulate the browser behavior while giving researchers the flexibility to experiment with underlying transport and application protocol implementations.

In this work, we introduce BrowsEm, an emulator for web application workloads that gives network researchers a flexible and adjustable tool to assess the impact of changes under realistic conditions while avoiding intractable complexity. For the emulation, we derive a model of the page loading process, capturing meta-data of HTTP transactions. The approach reduces the complexity of client and server web applications in that the actual content is not interpreted

or rendered, but replaced with timing and dependency information. We identify model parameters by extracting and storing information from the website loading behavior of web browsers and the network stack. This scraping process is fully automated. The resulting file is used by the emulator which replicates the loading behavior on the transport and HTTP layer without the need to set up complex infrastructure or to modify live websites.

The resulting emulation helps in understanding the impact of changed network path characteristics and modifications to the transport or application layer on the loading time of websites. Users can either modify model parameters, such as changing the HTTP version of requests, or experiment with client or server side protocol features, e. g., added new HTTP priority signals or multipath connections. The emulator is designed in a modular way such that different implementations on client and server side are possible. The evaluation shows that BrowsEm is capable of replicating realistic web workloads with remarkable accuracy, achieving a relative page load time (PLT) error of ± 0.25 for 80% of sampled sites.

The remainder of the paper is structured as follows: The next section provides an overview of related work. Section 3 defines the scope of the system model of BrowsEm. The implementation is explained in Section 4. In Section 5, we evaluate and discuss results of performed measurements. Finally, after explaining the limitations of the approach in Section 6, a conclusion summarizes the main findings.

The code of the tool is available online[1].

2 Related Work

The demand for the realistic and reproducible application of web traffic models dates back to the early versions of HTTP. Barford and Crovella [2] state that either analytical or trace-based approaches can be used to generate synthetic web workloads. Their *Scalable URL Reference Generator* (SURGE) tool was employed to investigate web performance in Internet-wide measurements, whose traffic adhered to mathematical models mimicking HTTP request properties of users interacting with a web page [1].

SURGE's analytical model has similarities with Mah's [10] and Choi and Limb's [7]. It accounts for statistically representative file and request sizes, their relative popularity to other server resources, re-request likelihood, how they are referenced in a page, reference count, and traffic burstiness. These values were derived from data sets [4] which empirically captured web user behavior. SURGE's synthetic traffic matched the variability of real traffic and put a more realistic load on servers than other benchmarking tools at the time. Per Barford and Crovella's definition, BrowsEm is trace-based as it emulates realistic workloads through the playback of pre-recorded samples. According to them, such an approach has the tendency to become a black box that obscures system behavior and is hard to adapt to new requirements. We believe to have addressed such shortcomings through the use of a flexible and configurable model format.

Noting significant web developments since the late 1990s, such as the rise of video streaming applications, Lee and Gupta [9] proposed a traffic generator recognizing that users may change pages before embedded objects are fully downloaded or be using multiple browsers at once. Pries et al. [15] compared Lee's, Chois', Barford's, and Ihm's [8] measurements with a traffic model based on the top million accessed web sites and identified the retrieval of large multimedia content spread across the globe as a trend. Their HTTP traffic model was experimentally integrated into the `ns-3` QUIC module [14]. Web page load times over HTTP/1.1, SPDY, and QUIC were evaluated with the Mahimahi framework [12], which can accurately record and replay HTTP traffic locally using fully isolated, composable UNIX shells that emulate multi-server applications and network conditions. The Mahimahi-supported finding that the serialization of requests underutilizes links led to improving HTTP application performance over long-delay links.

3 Scope of Emulation Model

The scope of the emulation model is shaped by the needs of protocol researchers and web site operators investigating the continuously evolving web stack.

Modern web browsers are sophisticated software systems. Their complexity is driven by web standards and the need to interpret, execute, render, and display multimedia content. When loading a website, the requested document often includes other assets, such as images, JavaScript, or stylesheets. If a referenced asset is hosted on a different web server, this can lead to information traveling on a different network path with different properties and additional overhead due to the establishment of a new connection.

In the networking-focused BrowsEm model, all interpretation of content is abstracted at the client and replaced with inter-transaction dependencies and timing information. The timing of individual HTTP transactions are further divided into several phases, namely *blocked*, *dns*, *connect*, *tls*, *send*, *wait*, and *receive*. The path between client and server is modeled with capacity, delay, and loss properties. Server-side web applications also introduce application limited phases. We model those by delaying HTTP responses by a specific time. Figure 1 depicts a simplified drawing of the model.

[1] https://github.com/holzingk/BrowsEm

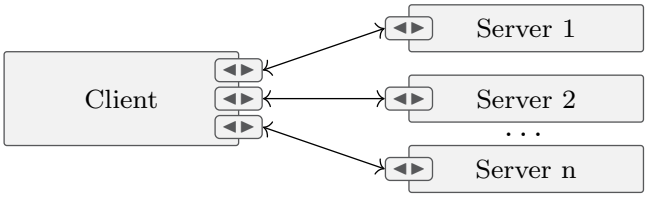


Figure 1: Simplified emulation model

4 Approach

The BrowsEm workflow consists of four main parts. To build realistic emulation models, we first derive model parameters by observing the page load process of a web browser, referred to as data scraping. Then, the collected data are postprocessed and converted into the workload data model file. Since BrowsEm users typically want to assess the impact of modifications in the web stack, manual edits can be made there. Moreover, alternate versions of components such as web servers or QUIC libraries can be deployed inside the emulator. The workload data model is subsequently provided as input to the page load emulation, which replicates the HTTP and transport layer workload. Finally, users can inspect the resulting performance metrics.

4.1 Data Scraping

In order to emulate page load behavior, sample data are needed. To be as realistic as possible, we chose to observe and record the behavior of a web browser while loading web sites and use the scraped data as input to the emulator. For now, our tool is limited to the Chromium browser but since the used interface is also offered by other browsers, we anticipate that they can be supported as well. To automate the recording of the page loading process, BrowsEm provides a Python tool. It extracts information from the browser and from traffic captures. It uses the Chrome DevTools Protocol (CDP)[2] and PyShark[3], a Python interface to interact with the Wireshark network packet analyzer. HTTP Archive (HAR) files, transaction dependencies, and traffic sniffing are available data sources. qlog, a structured logging format for QUIC, would be an additional option [11]. However, it is not supported by Chromium.

HAR. All major browsers have built-in developer tools that record network activity and page load times. A de-facto standard for those data are HAR files which include all requests the browser made while loading a page [13]. It is a structured JSON log, containing detailed subtimings for each resource: The *blocked* time of a request is the delay caused by the browser, e. g., due to a reached connection limit. Also, the duration of a potential DNS lookup is encoded (*dns*), along with the time for connection establishment (*connect*), duration of the Transport Layer Security (TLS) handshake (*tls*) and time to send the request to the server (*send*). In addition, the *wait* time indicates the time from the completed request to the first byte of the response, i. e., the time to first byte (TTFB). Finally, the *receive* time measures the time to complete the request. We use the CDP to initiate page loading with HAR file recording and write a TLS key log file, allowing to decrypt recorded packet traces.

Transaction dependencies. HAR files include initiator fields. They reference an other transaction that triggered this particular one, which we use to infer a transaction dependency tree. Although this information sometimes is missing, an initiator can clearly be identified through manual inspection. To address this problem, we created a dependency tree inference heuristic. We assume that child requests typically start shortly after the parent began receiving its response. We identify the parent transaction by finding the request with the smallest positive delta between its response start time and the child's request start time bounded by a specific threshold. Although by nature this approach is not free of errors, evaluation yielded sufficient accuracy.

Traffic sniffing. Because not all relevant data are included in the HAR file, some data are extracted from recorded traffic traces. PyShark decrypts TLS and QUIC packets with the key log file, enabling us to match relevant flows to HTTP requests and to provide additional insights into HTTP usage. We assess crucial path characteristics per server IP adress. We use several mechanisms to assess the path round-trip time (RTT). In the case of TCP, we use the median RTT obtained by matching the timestamp value and timestamp echo reply, if available. As fallbacks, in the following order, handshake timestamps or ICMP echo request and reply as estimate are used respectively. The QUIC spin bit is an additional RTT estimation source that is not yet used by the Chromium browser. Another option would be to track QUIC packet and TCP segment numbers and correlating acknowledgements.

For the path capacity, we integrate the passive estimation approach PPrate [5] for which an open-source implementation has been provided [3]. Some connections only exchange a limited number of packets, likely never leaving the slow start phase of congestion control. As a consequence, some results are erroneous.

4.2 Workload Data Model

The data model describes the emulated workload and is serialized as JSON. It combines input from the HAR file and analyzed sniffed traffic recorded during web scraping. The

[2]https://chromedevtools.github.io/devtools-protocol/
[3]http://kiminewt.github.io/pyshark/

main content is a list of HTTP transactions. Connection identifiers distinguish connections in case of multiplexed protocols such as HTTP/2 and HTTP/3. Absolute timestamps indicate the start and end times of the request and response. Relative request timings taken from the HAR file provide additional information. We append the request and response body length and the depending and dependant transaction identifiers obtained from the initiator field or dependency heuristic. The data model contains network path information in the form of loss percentage, RTT and estimated capacity.

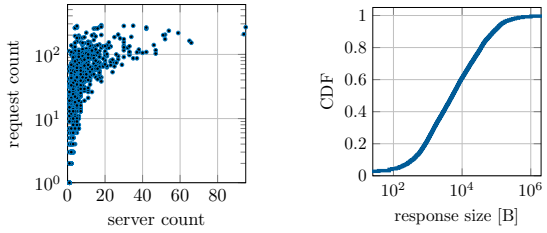
The structured file allows for easy modification of the workload model. Users, for instance, can change the HTTP version of individual requests, adjust RTT or bandwidth constraints, vary the number of parallel connections, or add additional requests. This makes it possible to simulate alternative versions of a web site's behavior under different web stack configurations, while still preserving the realism of the original workload.

4.3 Page Loading Emulation

The page loading emulation uses the Tokio[4] asynchronous Rust runtime. BrowsEm includes network path behavior in its data model. It uses Linux network namespaces in conjunction with the NetEm network emulator to imitate the behavior of distinct paths between the client and the servers on a single node [6].

For each separate server in the model, a dedicated Tokio task is spawned. The server is defined as an abstract behaviour using a Rust trait, so new or modified servers can be added by users that provide implementations for that interface. Currently, there are three example implementations. We implemented the trait, for one, using Actix Web[5] which is a Rust web framework that supports HTTP/1.0, HTTP/1.1, and HTTP/2. For another, using quinn as an HTTP/3 server. Last, we support using nginx as an example for a general purpose web server. The trait defines two asynchronous functions. The first is a request callback which takes the workload model as input to initialize the server, the second is a callback that handles the actual emulated requests.

The client uses libcurl which is a widely used and flexible HTTP client library. It was chosen because it provides a single API for several HTTP versions and also supports five QUIC implementations: Cloudflare quiche, ngtcp2, OpenSSL 3.2 QUIC, msquic and the experimental Linux Kernel QUIC. For our evaluation we use ngtcp2. As an initializing step, before the actual workload emulation is performed, the client sends the workload model to the servers. At the client, each connection is handled by a Tokio task which itself spawns tasks for the individual transactions which are started in accordance with the timing information found in the load model data. The *wait* time in the model data includes the network path RTT and additional delays caused by the server application. In the emulation, we subtract the RTT from the *wait* time, since we apply network path emulation, allowing to change network path characteristics. Before a request is made in the emulator, the dependency tree is respected by considering the delay from the finalization of the parent response to the request of the child. Depending on the model data, either a new connection to the server is initiated, or an existing connection is reused. Similar to most browsers, the TCP nodelay option is set.



(a) Request and server count per loaded web site **(b) HTTP response body sizes of all web sites**

Figure 2: Statistics of scraped sites

5 Evaluation

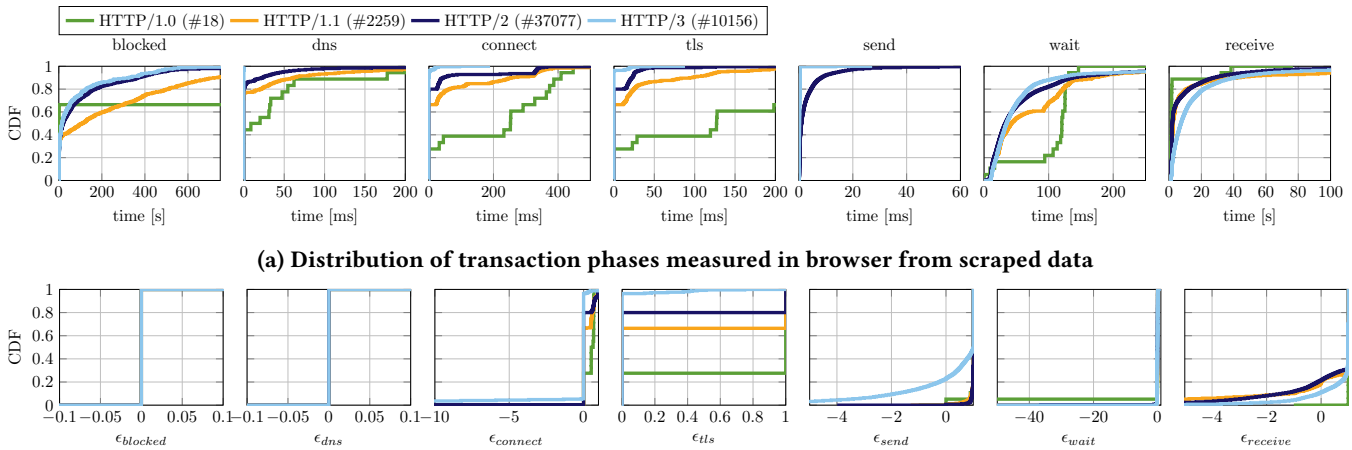
We start this section with a data driven motivation, advocating for the need of a tool such as BrowsEm. The primary goal of our evaluation is to quantify the error it introduces while reproducing the loading behavior of web sites. We compare emulation results to measurements from recorded browser sessions. In addition, we assess the model stability by modifying emulation parameters such as the RTT and investigate whether results are consistent with real-world page loads under similar conditions.

5.1 Motivation

To perform the evaluation, we use the top 1000 web sites from the DomCop List of Top 10 Million Domains[6] for scraping. Some web sites caused issues, e. g., due to the use of WebSockets which we do not yet support or because headless user agents are blocked. The following data are based on 922 sampled sites. We only record the loading process of the start page and do not perform any interactions with the site. We use a fast connection within a national science network.

In Figure 2a, we provide an overview of the number of servers involved and requests count for each page load. It can be seen that loading most web sites establishes connections to several different servers and make tens or hundreds of

[4]https://tokio.rs/
[5]https://actix.rs/
[6]https://www.domcop.com/top-10-million-domains



(a) Distribution of transaction phases measured in browser from scraped data

(b) Relative differences of transaction phases between measurements of real web sites and emulation

Figure 3: Duration of transaction phases as CDFs of all requests separated by HTTP version; different x-axis scales

requests. Figure 2b shows the distribution of response body sizes across all 49510 transactions in the scraped data set. The vast majority are transfers of smaller than 100 kB.

Figure 3a depicts cumulative distribution functions (CDFs) of the duration of all phases for each HTTP version. They include all requests extracted from HAR files generated while loading the 922 web sites. The dataset includes 18 HTTP/1.0, 2259 HTTP/1.1, 37077 HTTP/2, and 10156 HTTP/3 transactions. It shows that major contributors to the transaction duration are the *blocked*, *wait*, and *receive* times.

From these statistics, we conclude that web sites have a very mixed loading behavior, leading to a huge variety of workloads imposed on the HTTP and transport layer. To reproduce those outside of a web browser, while accounting for the timing of transaction phases, a tool such as BrowsEm is needed — a purely synthetic approach is not sufficient.

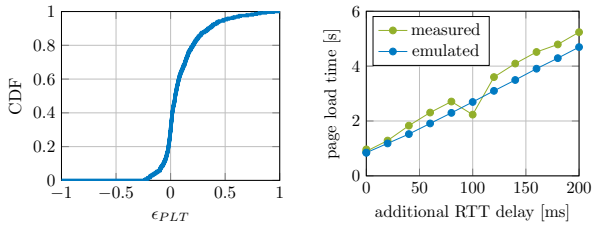
5.2 Metrics

To quantify the emulation quality of BrowsEm we introduce two metrics. For the first, we define the relative difference between the scraped page load time s_{PLT} and emulated page load time e_{PLT} as the PLT error $\epsilon_{PLT} = \frac{s_{PLT} - e_{PLT}}{s_{PLT}}$. It captures how well the overall loading behavior and request dependency structure are reproduced. The second metric, $\epsilon_P = \frac{s_P - e_P}{s_P}$, measures the relative difference of individual phases of HTTP transactions. It can help to identify the root cause of emulation deviations.

5.3 Emulation Error

Figure 3b shows relative differences of transaction phase durations ϵ_P over all observed transactions. For reference, see also Figure 3a, which shows the absolute distribution of scraped transaction phase times. The sample counts can be found in the legend of Figure 3. Due to a limited number of HTTP/1.0 transactions, related results need to be interpreted with care. The *blocked* time is always met by the tool, since as the name suggests, no operations are performed. Similarly, BrowsEm replaces DNS lookups with wait time. The shown *connect* times largely matches the browser behavior. There is a small fraction of HTTP/3 transactions with large relative differences. Since HTTP/3 transactions have a very short absolute *connect* time, we deem those outliers as insignificant. We model TLS and encryption for our servers as well. HTTP versions greater than HTTP/1.0 reuse connections, resulting in a large number of transaction with *tls* time of 0 ms. The emulation tends to complete the TLS handshake faster. Possible reasons for the deviations could be the use of different encryption libraries at the client and server compared to the real-world measurement, or influences of an erroneous RTT estimate. The absolute durations of the *send* time are typically very short. In the emulation it tends to be even shorter than in the browser. There, it could be affected by additional application or congestion limitations not covered in our model, hinting towards wrong path capacity estimations. The *wait* time is mostly met by BrowsEm. It is realized in the emulated server by waiting for the measured time, whereby the path RTT is deduced (see Section 4.3) — a possible cause of differences. The *receive* time, albeit being generally longer due to larger transfer sizes, shows similar behavior as the *send* time and the same explanations apply. In conclusion, while deviations from the observed browser timings exist, the emulation represents individual transaction phases reasonably well.

Next, we assess the relative difference of the PLT, shown in Figure 4a. The PLT metric ϵ_{PLT} indicates whether the loading behavior of a web page, particularly the dependency



(a) Distribution of relative differences between scraped and emulated total page load times

(b) Absolute differences of PLT between measurements of real web site and emulated scenario

Figure 4: Page load time and study of model robustness

tree of requests and their timings is accurately emulated. We find that about 80% page loads have less than $\pm 0.25 \epsilon_{PLT}$.

5.4 Stability

Using the model file, users can modify emulated request parameters such as path properties or the HTTP version. To evaluate the reliability of results, we assess the model stability by comparing those emulations against measurements taken directly in a browser under the same conditions.

For that purpose, we take the model derived from scraping a web site under baseline conditions, i. e., without introducing any additional delay. To study the effect of network latency, we run the emulation repeatedly, with manually increased path RTT properties, keeping all other aspects constant. For each configuration, we compare the emulated results against ground-truth measurements obtained from actual browser sessions under the same added delay. This allows us to evaluate the robustness of our emulation model.

The results of this experiment are shown in Figure 4b. We can observe that the overall PLT is steadily increasing with the added RTT, as expected. The emulated PLT closely follows the trend of the real browser measurements, indicating that our model is capable of accurately capturing the impact of network latency on web page loading times. In terms of the relative error ϵ_{PLT}, we can observe an increase with added RTT.

6 Limitations

Our model is inherently limited by which data is measureable in the browser and the network stack. Thus, we only implicitly capture browser-specific behaviors such as client-side rendering, JavaScript execution, or CSS rendering through the timing of requests and responses. Therefore, some web Quality of Experience (QoE) metrics like Time to First Paint are not captured, but could be explored in future work. The use of static network conditions limits realism, as real-world conditions vary significantly. Also, we do not directly cover application limited phases in neither server nor client. The network path estimation showed to be a source of errors. While request dependency modeling performs well in many cases, it may not fully capture the behavior of all web applications. Additionally, parts of the model rely on data reported by Chromium and may not generalize to other browsers such as Safari or Firefox, which use a different networking stack. Future work could enhance the model by adding more complexity, such as including priority signals and evaluating the impact on the accuracy of the emulation.

7 Conclusion

To improve the web network stack, protocol researchers, network engineers, and site operators are interested in measuring performance of web applications while having full flexibility of underlying implementation details, network parameters, and configurations. To achieve that, replicating deployments in reproducible testbed setups is tedious because the deployment, modification, and orchestration of involved browser and server software is complex. BrowsEm addresses this gap by providing a flexible and extensible measurement framework, capable of replicating network workloads of web applications with a focus on network path properties, transport and HTTP application layer.

The tool includes an automated data scraper to extract the workload model from browser behavior. We use HAR files and traffic traces as data sources. Gathered information is collected into a model file. It contains detailed information on the order and dependency of HTTP transactions and the timing of their individual phases as well as inferred network path properties such as delay and path capacity.

The emulator replicates the page loading process by replacing processing time spent with generating or interpreting web payload content with wait time. It strives to meet the duration of all relevant transaction phases. Based on the model information, network paths are emulated on a single node using Linux network namespaces. The client uses libcurl as a flexible and extensible HTTP library, which has multiple backends for HTTP/3. We provide three implementations for the server. All standard HTTP versions are supported.

We evaluate BrowsEm on a sample of 922 popular web sites. We compare the time difference of individual transaction phases between browser measurements and the emulation. Results show that, despite some deviations, the emulation approximates real behavior sufficiently well. The total PLT has less than ±0.25 relative error in 80% of emulated page loads. In an additional measurement campaign, the emulator was assessed for robustness and yielded good stability towards changed network path parameters.

The code of the tool is available online[1].

Acknowledgments

This work was supported by the EU Horizon Europe programme, projects SLICES-PP (10107977) and GreenDIGIT (101131207), by the German Federal Ministry of Education and Research (BMBF), projects 6G-life (16KISK002) and 6G-ANNA (16KISK107), by the German Research Foundation, project HyperNIC (CA595/13-1), and by the Bavarian Ministry of Economic Affairs, Regional Development and Energy, project 6G Future Lab Bavaria.

References

[1] Paul Barford and Mark Crovella. 1999. Measuring Web performance in the wide area. 27, 2 (1999), 37–48. doi:10.1145/332944.332953

[2] Paul Barford and Mark Crovella. 1998. Generating representative Web workloads for network and server performance evaluation. In *Proceedings of the 1998 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems* (Madison, Wisconsin, USA) *(SIGMETRICS '98/PERFORMANCE '98)*. Association for Computing Machinery, New York, NY, USA, 151–160. doi:10.1145/277851.277897

[3] Simon Bauer, Janluka Janelidze, Benedikt Jaeger, Patrick Sattler, Patrick Brzoza, and Georg Carle. 2023. On the Accuracy of Active Capacity Estimation in the Internet. In *2023 IEEE/IFIP Network Operations and Management Symposium (NOMS 2023)*. Miami, USA.

[4] Carlos R Cunha, Azer Bestavros, and Mark E Crovella. 1995. *Characteristics of WWW client-based traces.* Technical Report. Boston University Computer Science Department.

[5] Taoufik En-Najjary and Guillaume Urvoy-Keller. 2006. Pprate: A passive capacity estimation tool. In *2006 4th IEEE/IFIP Workshop on End-to-End Monitoring Techniques and Services*. IEEE, 82–89.

[6] Stephen Hemminger et al. 2005. Network emulation with NetEm. In *Linux Conf AU*, Vol. 5. 2005.

[7] Hyoung-Kee Choi and J.O. Limb. 1999. A behavioral model of Web traffic. In *Proceedings. Seventh International Conference on Network Protocols* (Toronto, Ont., Canada, 1999). IEEE Comput. Soc, 327–334. doi:10.1109/ICNP.1999.801961

[8] Sunghwan Ihm and Vivek S Pai. 2006. Towards understanding modern web traffic. (2006).

[9] Jeongeun Lee, Maruti Gupta, and Intellon Corp. 2007. A NEW TRAFFIC MODEL FOR CURRENT USER WEB BROWSING BEHAVIOR. https://api.semanticscholar.org/CorpusID:14057598

[10] Bruce A. Mah. 1997. An empirical model of HTTP network traffic. In *Proceedings of INFOCOM '97*, Vol. 2. 592–600 vol.2. doi:10.1109/INFCOM.1997.644510

[11] Robin Marx, Luca Niccolini, Marten Seemann, and Lucas Pardue. 2025. *QUIC event definitions for qlog*. Internet-Draft draft-ietf-quic-qlog-quic-events-10. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-ietf-quic-qlog-quic-events/10/ Work in Progress.

[12] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, and Hari Balakrishnan. 2015. Mahimahi: Accurate Record-and-Replay for HTTP. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*.

[13] Jan Odvarko, Arvind Jain, and Andy Davies. 2012. HTTP Archive (HAR) Format. https://w3c.github.io/web-performance/specs/HAR/Overview.html

[14] Umberto Paro, Federico Chiariotti, Anay Ajit Deshpande, Michele Polese, Andrea Zanella, and Michele Zorzi. 2020. Extending the ns-3 QUIC Module. In *Proceedings of the 23rd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems* (Alicante Spain, 2020-11-16). ACM, 19–26. doi:10.1145/3416010.3423224

[15] Rastin Pries, Zsolt Magyari, and Phuoc Tran-Gia. 2012. An HTTP web traffic model based on the top one million visited web pages. In *Proceedings of the 8th Euro-NF Conference on Next Generation Internet NGI 2012* (Karlskrona, Sweden, 2012-06). IEEE, 133–139. doi:10.1109/NGI.2012.6252145