Mahimahi: A Lightweight Toolkit for Reproducible Web Measurement

Ravi Netravali, Anirudh Sivaraman, Keith Winstein,
Somak Das, Ameesh Goyal, and Hari Balakrishnan
MIT Computer Science and Artificial Intelligence Laboratory
{ravinet, anirudh, keithw, somakrdas, ameesh, hari}@csail.mit.edu

ABSTRACT

This demo presents a measurement toolkit, Mahimahi, that records websites and replays them under emulated network conditions. Mahimahi is structured as a set of arbitrarily composable UNIX shells. It includes two shells to record and replay Web pages, *RecordShell* and *ReplayShell*, as well as two shells for network emulation, *DelayShell* and *LinkShell*. In addition, Mahimahi includes a corpus of recorded websites along with benchmark results and link traces (https://github.com/ravinet/sites).

Mahimahi improves on prior record-and-replay frameworks in three ways. First, it preserves the multi-origin nature of Web pages, present in approximately 98% of the Alexa U.S. Top 500, when replaying. Second, Mahimahi isolates its own network traffic, allowing multiple instances to run concurrently with no impact on the host machine and collected measurements. Finally, Mahimahi is not inherently tied to browsers and can be used to evaluate many different applications.

A demo of Mahimahi recording and replaying a Web page over an emulated link can be found at http://youtu.be/vytwDKBA-8s. The source code and instructions to use Mahimahi are available at http://mahimahi.mit.edu/.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Measurement Techniques; H.3.5 [**Information Storage and Retrieval**]: Online Information Services

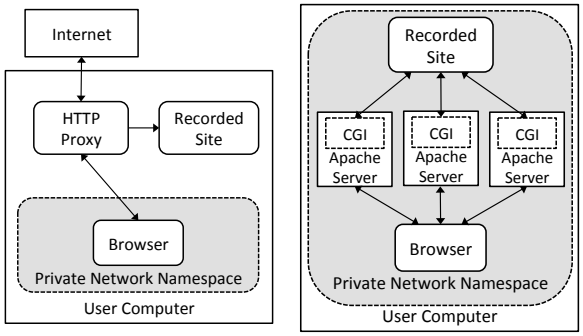
Keywords

Page Load Time; Record-and-Replay; Web Measurements

1. INTRODUCTION

Mahimahi is a toolkit that can be used to evaluate how effective techniques that aim to make the Web faster perform over different network conditions. This question is of interest to network protocol designers who seek to understand the application-level impact of new multiplexing protocols, Web developers who wish to speed up access to their websites, and browser developers who need to evaluate how changes to their DOM and JavaScript parsers impact page load times. Mahimahi is structured as a set of UNIX shells, each of which we describe below.

SIGCOMM'14, August 17–22, 2014, Chicago, IL, USA.
ACM 978-1-4503-2836-4/14/08.
http://dx.doi.org/10.1145/2619239.2631455.



(a) RecordShell (b) ReplayShell

Figure 1: Arrows indicate direction of traffic

2. MAHIMAHI'S SHELLS

RecordShell. RecordShell (Figure 1a) records HTTP data sent during actual page loads and stores it on disk for subsequent replay. RecordShell spawns a man-in-the-middle proxy, equipped with an HTTP parser, on the host machine to store and forward all HTTP(S) traffic both to and from an application running within RecordShell.

At the end of a page load, a recorded folder contains a file for each request-response pair seen during that record session. RecordShell is compatible with any unmodified browser because recording is done transparently without modifying browser settings.

ReplayShell. ReplayShell (Figure 1b) mirrors a website using content recorded by RecordShell. ReplayShell accurately emulates the multi-origin nature of websites by spawning an Apache 2.4.6 Web server for each distinct IP/port pair seen while recording.

To operate transparently, ReplayShell binds its Apache Web servers to the same IP address and port number as their recorded counterparts. ReplayShell creates a separate virtual interface for each distinct server IP. All browser requests are handled by one of ReplayShell's servers, each of which can access the entire recorded content for the site. The Apache configuration redirects incoming requests to a CGI script which compares each request to the set of all recorded request-response pairs to locate a matching response.

DelayShell. DelayShell emulates a link with a fixed minimum one-way delay. All packets to and from an application running inside DelayShell are stored in a packet queue. A separate queue is maintained for packets traversing the link in each direction. Each packet is released from the queue after the user-specified one-way delay, enforcing a fixed per-packet delay.

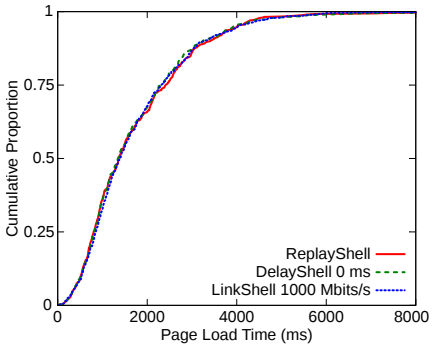


Figure 2: DelayShell's and LinkShell's low overhead

	Machine 1	Machine 2
CNBC	7584 ± 120 ms	7612 ± 111 ms
wikiHow	4804 ± 37 ms	4800 ± 37 ms

Table 1: Mean $\pm$ standard deviation for page load times across machines

LinkShell. LinkShell is used to emulate a link using packet-delivery traces. It is flexible enough to emulate both time-varying links such as cellular links and links with a fixed link speed. When a packet arrives into the link, it is directly placed into either the uplink or downlink packet queue. LinkShell releases packets from each queue based on the corresponding packet-delivery trace. Each line in the trace is a packet-delivery opportunity: the time at which an MTU-sized packet will be delivered in the emulation.

3. SUITABILITY FOR ACCURATE MEASUREMENT

Low overhead. Mahimahi imposes low overhead on page load time measurements. We illustrate this in Figure 2, which shows that when loading the 500 websites in our corpus, DelayShell with 0 ms imposes only a 0.15% overhead on median page load time compared to ReplayShell alone, while LinkShell with a 1000 Mbits/s trace adds 1.5% on top of ReplayShell.

Reproducibility. Table 1 shows a summary of the distribution of page load times when loading two Web pages, www.cnbc.com and www.wikihow.com, 100 times each on two separate host machines. The mean page load times for each site are less than 0.5% apart across the two machines, suggesting that Mahimahi produces comparable results on different host machines. Similarly, the standard deviations are all within 1.6% of their means, suggesting that Mahimahi produces consistent results on a single host machine.

4. NOVELTY

This section describes several new features in Mahimahi as compared with existing record-and-replay tools such as Google's web-page-replay [1].

Multi-origin Web pages. Unlike other tools, ReplayShell preserves the *multi-origin* nature of websites: websites today commonly include content belonging to several distinct origin servers. As we show below, preserving the multi-origin nature of websites is critical to the accurate measurement of page load times.

A non-trivial number of websites today are multi-origin. Using our corpus of recorded sites, we computed the distribution of physical servers per website in the Alexa U.S. Top 500. The median number of servers is 20 while the 95th percentile is 51. Only 9 Web pages use a single server.

To evaluate the impact of not capturing the multi-origin nature of websites, we modify ReplayShell to serve all content from a single Web server. Table 2 shows the median and 95th percentile difference in page load time between when multi-origin nature is and is not preserved, over nine different network configurations. Although the page load times are comparable over a 1 Mbit/s link, not capturing the multi-origin nature yields significantly worse performance at higher link speeds.

	30 ms	120 ms	300 ms
1 Mbit/s	1.6%, 27.6%	1.7%, 10.8%	2.1%, 9.7%
14 Mbits/s	19.3%, 127.3%	6.2%, 42.4%	3.3%, 20.3%
25 Mbits/s	21.4%, 111.6%	6.3%, 51.8%	2.6%, 15.0%

Table 2: 50th, 95th percentile page load time difference without multi-origin preservation

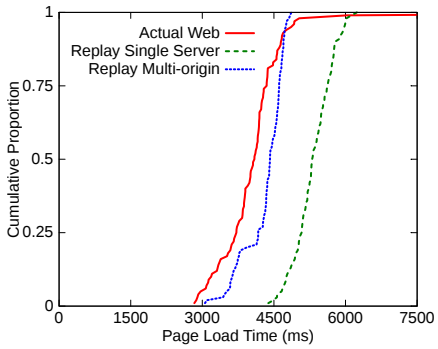


Figure 3: Multi-origin preservation yields measurements closer to the Web

We further illustrate the importance of multi-origin preservation by comparing measurements collected using ReplayShell to real page load times on the Web. Figure 3 shows the distribution of page load times when loading www.nytimes.com 100 times on the Web and inside ReplayShell with and without multi-origin preservation. For fair comparison, we record the minimum round trip time to www.nytimes.com for each page load on the Web and use DelayShell to emulate this for each page load with ReplayShell.

ReplayShell with multi-origin preservation yields page load times that more accurately resemble page load times collected on the Internet. The median page load time is 7.9% larger than the Internet measurements, which is less than the 29.6% discrepancy when the multi-origin nature is not preserved.

Isolation. Each namespace created by Mahimahi is separate from the host machine's default namespace and every other namespace. As a result, processes running inside the namespace of a Mahimahi tool are completely isolated from those running outside. This means that host machine traffic does not affect the measurements reported by Mahimahi, and Mahimahi's network emulation does not affect traffic outside of Mahimahi's network namespaces. This enables many different configurations to be concurrently tested on the same host machine, and in complete isolation from each other. In contrast, web-page-replay modifies DNS resolution on the host machine and affects all traffic from the host machine.

Beyond browsers. Although most existing record-and-replay frameworks only replay browser page loads, Mahimahi's design allows it to replay any application that uses HTTP. For instance, a mobile device emulator, such as for Android [2], can be used to analyze and measure mobile application performance.

References

[1] http://code.google.com/p/web-page-replay.
[2] http://developer.android.com/tools/devices/emulator.html.