Measuring and Injecting Latency in Web Apps

Adam Goode, Steven Hillenius, Bonnie John, M. Satyanarayanan

June 2009
CMU-CS-09-142

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

In mobile computing and other contexts, latency is a more critical resource than bandwidth. Interactive performance relies on low latency response to the user. The question of how much performance is necessary (and when) is still open.

These sources of latency manifest themselves in different ways and with differing severities. When designing systems for a Web 2.0 framework, it is important to have a good means for measuring existing latencies as well as simulating new latencies, throughout all parts of a system.

This report discusses a web app latency analysis framework with respect to a set of representative tasks. These tasks include: web usage, navigation with tabs, navigation between many open windows, and significant copying and pasting.

This research was supported by the National Science Foundation (NSF) under grant number CNS-0833882. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF or Carnegie Mellon University.

Keywords: latency, logging, network simulation, web 2.0, web applications

1 Motivation and Background

In mobile computing and other contexts, latency is a more critical resource than bandwidth. Interactive performance relies on low latency response to the user. The question of how much performance is necessary (and when) is still open.

Previous work looked at the effects of timesharing systems on users. More recently, there has been a focus on interactive performance of web pages, for example Galletta looked at the effects of delay on user performance, attitudes, and intentions [6]. Since the time of Galletta's work, a new kind of interaction has emerged on the web, combining local processing with background server communication. Various names for this new interaction method include Web 2.0, AJAX, and Rich Internet applications.

In Web 2.0 applications, there are more potential sources of latency than before. These include:

- Local processing delay
- Remote (server-side) delay
- Network delay

These sources of latency manifest themselves in different ways and with differing severities. When designing systems for a Web 2.0 framework, it is important to have a good means for measuring existing latencies as well as simulating new latencies, throughout all parts of a system.

In this report, we will be looking at applying our framework to a modern version of Card, Moran, and Newell's Experiment 8A "Experimental Validation of the Model" [2, p. 270]. The tasks include significant web usage, navigation with tabs, navigation between many open windows, and significant copying and pasting. For the purposes of validation (section 4), we will limit our task to the creation of a two-hour long event in Google Calendar (`http://www.google.com/calendar/`). This involves clicking, dragging, and typing.

The idea of running a new version of Experiment 8A came about after attempting to build a cognitive model of a user performing a task with a modern interface. This user leveraged extensive keyboard shortcuts to navigate through websites and enter text, and the model was predicting much slower performance for the task than was observed directly. After verification of the cognitive modeling system (which is based in part on results from Experiment 8A), it was hypothesized that the experimental results generated in 1983 do not accurately predict the performance of today's computer users.

This new version of the experiment was built in 2008, and although the experiment was up and running, no logging system were accurate enough for our needs. In some situations, the logger slowed down the computer so much that it nearly doubled pilot experiment times. Our ideal goal was to have a logger that was as good or better to what Card, Moran, and Newell used in their verification experiments in 1983. (The original experiment specified a logging tolerance of 33 ms. [2, p. 154]) Until we put together the system described in this report, we had not found an accurate logging system, and could not run our experiment.

2 Existing Tools

For this project, we first evaluated several tools for both measuring and injecting latency. Some tools were useful with some modifications, and others were usable directly.

2.1 Measuring Latency

Latency can be measured by taking the time difference between user action and system response. For our work, we wanted to record keyboard and mouse action at a high temporal resolution (within 10 ms of jitter). This level of resolution is required because interesting events in human cognition happen at this timescale.

Initially, we wanted to be able to measure latency on both Windows and Mac OS systems. For this reason, we looked first at RUI [8] and VIA [7].

2.1.1 RUI (Penn State)

RUI [8] (Recording User Input), available at `http://acs.ist.psu.edu/projects/RUI/`, was developed for Windows and Mac OS to record and playback keyboard and mouse inputs.

2.1.2 VIA (Rensselaer)

VIA [7] (Visualization-Interaction Architecture), available at `http://www.cogsci.rpi.edu/cogworks/?view=modules.research.spec&id=63`, is a network-based mechanism for logging. A small piece of code is written for each platform under examination which connects to a logging server over TCP/IP. Its goals are similar to RUI: being a powerful tool for logging human interaction for research.

2.1.3 X11 + Record Extension

Although initially we wanted to have a logging system that worked on Windows and Mac OS, we realized that it was possible to relax this constraint. Because we were interested in testing user interaction with a web application (Google Calendar), we could do just as well by using the standard Firefox browser on a Linux based system with the X11 windowing system. The flexibility of this system allows us to configure the mouse and keyboard to work as they do on either the Mac or PC: an important consideration for users even when using a cross-platform application like a browser.

Once we started using X11, we could leverage the standard Record Extension[14] that allows us to record mouse and keyboard events with very low jitter. By using the standard X server, Firefox, and unmodified Xnee[13] (a Record client), we were able to capture system-wide keyboard and mouse events with high fidelity. Additionally, a separate program was created to capture the text of the title of the active window as it changed while the user interacted with the system. Most importantly, all three kinds of data (mouse, keyboard, and title) are captured as a single stream of events with the same clock reference. This allows for easy reconstruction of events during later analysis.

Another tool we looked at was PyKeylogger[5]. While nicer than Xnee in some ways, it did not capture exact timings for keystrokes and mouse movements. Though it would be possible to customize PyKeylogger to work as we needed, Xnee worked well enough for now that we did not spend the effort of customization at this time.

2.1.4 Wireshark

Wireshark[3] is a program that captures and analyzes network packets. It is useful in logging network traffic that goes between the local machine and any remote servers. Sophisticated packet filtering and analysis are available. Though we didn't use this advanced functionality yet, it is likely to be useful in the future.

2.2 Injecting Latency

The second half of the system involves injecting latency into the system. This allows us to test psychological hypotheses related to user reactions to varying system response times. Because latency in different parts of the system will manifest in different ways, we need to be able to inject latency in various places.

2.2.1 Application Level

To inject latency at the application level, one must intercept and delay events coming from the user or delay the display of information going to the user.

In an existing study [6], we found that latency was injected using custom webpages with a delay function written in JavaScript. This works when an experiment (like Galletta's) has full control over all content, but it doesn't work when we do not control the server nor the pages (like Google Calendar). In the next section, we discuss our solution to this issue.

2.2.2 Network Level

There are many ways of injecting latency at the network level, both in software and in hardware. We will look at two existing tools that we are familiar with: NetEm[10] and the Apposite Technologies Linktropy 4500[1]. Both work on the same principle, by slowing or limiting network packets, network latency can be emulated. The difference is in the implementation: NetEm is software, embedded in the Linux kernel, and the Linktropy is a hardware device that sits standalone in a rack.

With NetEm, the standard way of injecting latency is a command like this: `tc qdisc add dev eth0 root netem delay 100ms`. This will introduce an outgoing delay of 100 ms to the network device `eth0`. Much more complicated setups are available, including introducing packet loss, duplication, and reordering. All of these events are generated by customizable statistical distributions, which allow for very realistic simulations. NetEm does have issues with simulation of newer radio-based networks (UMTS, WiMax, and others), but work is ongoing to address this[4].

The Linktropy 4500 basically encapsulates the functionality of NetEm into a hardware device with a web-based simple user interface. Either of these tools can work to inject latency, though

NetEm may be more useful for rapidly and automatically changing parameters (for example, between experimental cases), and for making per-packet decisions about how to apply emulation settings.

3 Our Tools

As mentioned in the previous section, we initially started with the idea of using cross-platform tools for logging. Since the experiment we are building is web-based, we were able to relax this requirement and use a Linux system running X11. This gave us more flexibility in logging and injecting latencies.

3.1 Keyboard, Mouse, and Titlebar Logging

For keyboard and mouse logging, we settled on using Xnee with the X Record Extension. Our target machine was a Fedora 10 system on a modern ThinkPad. One important issue with using this setup was that, by default, Fedora disables the Record Extension in X11.[1] It was necessary to recompile the X Server to enable this functionality. Once this was done, Xnee worked perfectly.

Below is an example trace log using our tools. This output was generated using Xnee and a custom X client derived from the `xev` tool. For the lines that do not start with a timestamp, the 8th field is the timestamp. Ignore fields that are not documented below:

Starts with	**Meaning**
`0,2`	Key down, 6th value is keycode given by `xmodmap -pk`
`0,3`	Key up, 6th value is keycode given by `xmodmap -pk`
`0,4`	Mouse down, 5th value is mouse button
`0,5`	Mouse up, 5th value is mouse button
`0,6`	Mouse motion: next two values are (x, y) position
[*timestamp*]	Active window (given by hex value) or titlebar has changed

In the lines with commas, the numbers given by the last column are in milliseconds, and represent the X server's clock. This number is typically the number of milliseconds since the kernel was booted. This is the output from Xnee. In the lines that start with a bracket, the timestamp comes first and is followed by the active window's id and title. This output is from our custom tool.

The log below shows a user navigating from the terminal to a Firefox browser. The user enters `www.cmu.edu` in the address bar, hits enter, and clicks on a link. Finally, the user closes the tab. Some mouse motion is elided for brevity.

```
0,6,664,537,0,0,0,1041338
0,6,665,537,0,0,0,1041350
0,6,666,537,0,0,0,1041386
[1041667] 0x2c00007 agoode@localhost:~/xnee-3.02
0,6,662,533,0,0,0,1041772
```

[1] This issue is tracked in Red Hat Bugzilla as `https://bugzilla.redhat.com/show_bug.cgi?id=472168`.

```
0,6,661,532,0,0,0,1041785
0,6,657,528,0,0,0,1041802
0,6,649,520,0,0,0,1041816
0,6,648,519,0,0,0,1041831
```

... mouse elided ...

```
0,6,667,347,0,0,0,1043375
0,6,679,347,0,0,0,1043392
0,6,701,349,0,0,0,1043407
```

... mouse is over Firefox window, note the active window event below ...

```
[1043408] 0x4600121 Mozilla Firefox
0,6,711,351,0,0,0,1043423
0,6,729,355,0,0,0,1043440
0,6,737,357,0,0,0,1043449
0,6,743,359,0,0,0,1043462
0,6,744,359,0,0,0,1043527
```

... user clicks to bring Firefox to the front, note the mouse down, mouse up ...

```
0,4,0,0,1,0,0,1043552
0,5,0,0,1,0,0,1043625
0,6,743,359,0,0,0,1043764
0,6,743,358,0,0,0,1043789
0,6,741,357,0,0,0,1043802
0,6,735,349,0,0,0,1043813
0,6,719,327,0,0,0,1043833
0,6,709,307,0,0,0,1043847
```

... mouse elided ...

```
0,6,615,157,0,0,0,1044800
0,6,615,155,0,0,0,1044830
0,6,614,155,0,0,0,1044845
0,6,614,154,0,0,0,1044858
```

... user clicks on address bar (mouse down, mouse up) ...

```
0,4,0,0,1,0,0,1045009
0,6,614,155,0,0,0,1045039
0,5,0,0,1,0,0,1045097
0,6,614,154,0,0,0,1045147
0,6,614,153,0,0,0,1045275
```

... user types “www.cmu.edu” and presses enter (note the key down, key up events) ...

```
0,2,0,0,0,25,0,1045693
0,3,0,0,0,25,0,1045757
0,2,0,0,0,25,0,1045851
0,3,0,0,0,25,0,1045907
0,2,0,0,0,25,0,1046010
0,3,0,0,0,25,0,1046089
0,2,0,0,0,60,0,1046146
0,3,0,0,0,60,0,1046295
0,2,0,0,0,54,0,1046296
0,2,0,0,0,58,0,1046391
0,3,0,0,0,54,0,1046429
0,3,0,0,0,58,0,1046488
0,2,0,0,0,30,0,1046540
0,3,0,0,0,30,0,1046619
0,2,0,0,0,60,0,1046755
0,3,0,0,0,60,0,1046843
0,2,0,0,0,26,0,1046879
0,2,0,0,0,40,0,1046935
0,3,0,0,0,26,0,1047014
0,2,0,0,0,30,0,1047038
0,3,0,0,0,40,0,1047076
0,3,0,0,0,30,0,1047148
0,2,0,0,0,36,0,1047229
0,3,0,0,0,36,0,1047317
[1047911] 0x4600121 Carnegie Mellon University - Mozilla Firefox
0,6,614,154,0,0,0,1048341
0,6,613,154,0,0,0,1048450
0,6,613,155,0,0,0,1048529
0,6,612,155,0,0,0,1049238
0,6,611,155,0,0,0,1049263
0,6,610,155,0,0,0,1049265
0,6,608,156,0,0,0,1049282
0,6,600,156,0,0,0,1049292
0,6,597,156,0,0,0,1049309
0,6,594,156,0,0,0,1049323
```

... mouse elided ...

```
0,6,473,684,0,0,0,1056114
0,6,474,683,0,0,0,1056154
0,6,475,683,0,0,0,1056177
0,6,475,682,0,0,0,1056188
0,6,476,681,0,0,0,1056203
0,6,476,680,0,0,0,1056218
0,6,477,680,0,0,0,1056231
```

```
0,6,476,680,0,0,0,1056890
0,6,475,680,0,0,0,1057088
0,6,474,680,0,0,0,1057226
```

... user clicks on a link ...

```
0,4,0,0,1,0,0,1057903
0,5,0,0,1,0,0,1057984
0,6,474,681,0,0,0,1057995
[1058634] 0x4600121 Alice 3 Software - Carnegie Mellon University - Mozilla
0,6,474,680,0,0,0,1060027
0,6,475,679,0,0,0,1060057
0,6,477,671,0,0,0,1060089
0,6,478,669,0,0,0,1060103
0,6,480,657,0,0,0,1060116
0,6,484,647,0,0,0,1060130
0,6,488,623,0,0,0,1060143
0,6,490,607,0,0,0,1060158
0,6,490,593,0,0,0,1060175
0,6,494,563,0,0,0,1060186
```

... mouse elided ...

```
0,6,362,249,0,0,0,1061363
0,6,362,250,0,0,0,1061375
0,6,363,251,0,0,0,1061575
0,6,363,252,0,0,0,1061596
0,6,364,253,0,0,0,1061615
0,6,364,254,0,0,0,1061703
0,6,364,255,0,0,0,1061871
```

... user clicks close tab button ...

```
0,4,0,0,1,0,0,1061962
0,6,365,255,0,0,0,1061972
0,5,0,0,1,0,0,1062037
[1062195] 0x4600121 Mozilla Firefox
0,6,365,256,0,0,0,1062659
0,6,367,256,0,0,0,1062671
0,6,369,257,0,0,0,1062684
```

It is with this logging output that we can record and analyze user actions in an experiment.

3.2 Application Level Delay and Logging

To inject delays into web apps, we implemented a simple (but flawed) JavaScript using the Greasemonkey[9] framework. The script injects itself into any webpage loaded in the browser and runs JavaScript code whenever a key is pressed or the mouse is clicked. In theory, with this mechanism, any event can be delayed on its way from the user to a web application. Unfortunately, JavaScript provides no current way for a function to synchronously delay execution. Workarounds exist, but frequently fail. Although this mechanism worked sometimes, it often halted the browser, breaking the interaction with the user. Another scheme must be found if application level delay is required.

The Greasemonkey script also logged the events it delayed by connecting to a logging server running on the local machine. Because Greasemonkey scripts cannot access the filesystem, logging had to be done via the `GM_xmlhttpRequest` method. A simple logging server implemented in Python was written to listen for these log events.

3.3 Network Level Delay

To introduce network delays, we used NetEm (as described in section 2.2.2) running on the local machine. With this setup, we could rapidly change the delay parameters as often as necessary, and could even selectively delay some packets and not others.

3.4 Network Logging

To log packets, we used Wireshark (as described in section 2.1.4).

4 Validation

We did not do any kind of analysis on latency injection. For latency measurement, we performed validations with the help of a special microphone, a video camera, and software called ELAN[12].

4.1 Experimental Setup

In order to verify the quality of each logger, we decided to measure the sounds of keystrokes with a sensitive microphone and then correlate the waveforms with the output of the logger.

The laptop is a ThinkPad T61. The microphone is a suction cup type, specifically a Peterson Signalflex SF20 Guitar Tuner Pickup microphone (see Figure 1). The microphone is placed on the trackpad of the laptop, which provides excellent signal from the sounds of keypresses and mouse click (see Figure 2). A video camera is mounted on a tripod and is aimed at the laptop. The microphone connects to the video camera through a cord.

4.2 Analysis

The software used for the validation is ELAN[12]. Its intended use is to annotate videos for language processing. In our case, we use it to mark the peaks of the key press/release or mouse

Figure 1: The Peterson Signalflex FL20 Guitar Tuner Pickup.

Figure 2: Closeup of the ThinkPad T61 with attached trackpad microphone.

click/release and then note what was pressed, was it a press or release, and what task was currently being undertaken by the user. (See figure 3.)

ELAN makes it very easy to move carefully through the video and audio to make sure that the annotation was placed correctly. Although video and audio are both captured, the video does not have enough temporal resolution as audio (NTSC video runs at 60 Hz, a frame every 16.67 ms), and is only used as a guide. All analysis is done exclusively with audio. Once annotation is complete, the annotations are exported from ELAN in CSV format.

After the data is exported, a "zero point" event is identified that matches a data point in both the logging data and the ELAN annotation. The difference is then calculated from all subsequent points. It is computed by subtracting the ELAN time from the logger time. Absolute difference is also computed. From there, the mean difference and mean of absolute differences are computed. An example of this data for Xnee is shown in table 1. RUI 1.0 is shown in table 2 and RUI 2.03 is shown in table 3. The zero point (shown in line 3 for Xnee, line 9 for RUI 1.0, and line 1 for RUI 2.03) is thrown away and not used in averages or any other calculations for the validation.

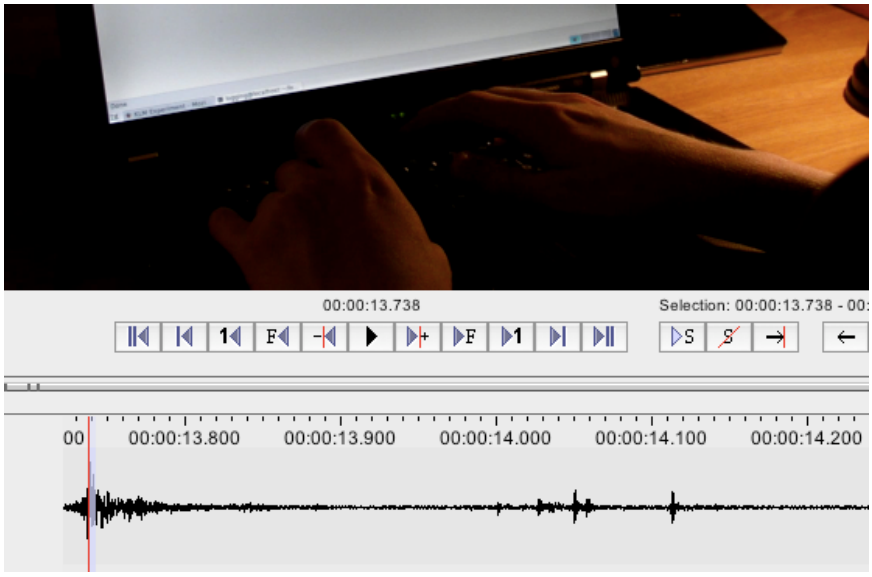

Figure 3: Audio and video in the ELAN interface.

Event type	Xnee time (ms)	ELAN time (ms)	Diff. (ms)	Abs. Diff. (ms)
Mouse down	−2865	−2863	−2	2
Mouse up	−2724	−2730	6	6
Key down	0	0	0	0
Key down	234	229	5	5
Mouse down	2567	2688	−121	121
Mouse up	2687	2708	−21	21
Mouse down	3996	3987	9	9
Mouse up	4155	4149	6	6
Mouse down	7085	7073	12	12
Mouse up	7225	7179	46	46
Mouse down	8235	8225	10	10
Mouse up	8365	8365	0	0

Table 1: Data for verification of Xnee. Because marking points in ELAN is a tedious manual process, only 12 points are identified (including the zero point).

4.3 Results

Analysis was done on three loggers: RUI 2.03, RUI 1.0, and Xnee.[2] The results are summarized in Table 4. For each observed event, a time difference is computed. For each logger, the mean of differences and the mean of absolute differences is shown in the results.

[2]VIA slowed down interaction with the interface so much that even if the logging were accurate, it would not be suitable for obtaining natural interaction patterns with an interface. Therefore, we did not analyze the accuracy of the VIA logger.

Event type	RUI 1.0 time (ms)	ELAN time (ms)	Diff. (ms)	Abs. Diff. (ms)
Key down	12499	12490	9	9
Key down	12733	12730	3	3
Mouse down	22186	22320	−134	134
Mouse down	33983	34130	−147	147
Mouse down	47983	48170	−187	187
Mouse down	71108	71260	−152	152
Mouse down	73842	74000	−158	158
Mouse down	82936	83100	−164	164
Key down	84030	84030	0	0
Key down	84358	84350	8	8
Key down	85217	85210	7	7
Mouse down	86936	87080	−144	144
Key down	87670	87710	−40	40
Key down	87967	87950	17	17
Mouse down	100327	100500	−173	173
Mouse down	102749	102920	−171	171
Mouse down	109545	109700	−155	155
Mouse down	117499	117670	−171	171
Mouse down	122186	118020	4166	4166
Mouse down	127592	127740	−148	148
Key down	131108	131270	−162	162
Key down	131264	131350	−86	86

Table 2: Data for verification of RUI 1.0.

Event type	RUI 2.03 time (ms)	ELAN time (ms)	Diff. (ms)	Abs. Diff. (ms)
Mouse down	22226	22226	0	0
Key down	25461	25508	−47	47
Mouse down	27206	26666	540	540
Mouse down	28023	28034	−11	11
Mouse down	30742	30756	−14	14
Mouse down	31539	31428	111	111
Mouse down	32882	32778	104	104
Key down	34711	34770	−59	59
Mouse down	36366	35694	672	672
Mouse down	36757	36630	127	127
Mouse down	37226	37242	−16	16
Mouse down	38736	38598	138	138
Mouse down	40054	39972	82	82

Table 3: Data for verification of RUI 2.03.

Logger	Mean Difference (ms)	Mean of Absolute Differences (ms)	# Events
RUI 2.03	136.00	160.00	12
RUI 1.0 PC	92.00	291.00	21
VIA	N/A	N/A	N/A
Xnee	−4.55	21.64	11

Table 4: Results of analysis of four loggers. VIA has no results because its method of data collection wasn't optimally suited for our task.

We can see that the Xnee solution is an order of magnitude more precise in measuring user events than other tools, and gets closer to the tolerance of the original experiment: "Accuracy of time-stamping was to within 33 msec of the actual time of event at the terminal."[2, p. 154] While our system is well on the way to being a useful tool, it still needs some work to reach the level of accuracy seen in 1983. Ensuring the use of high-speed timers and applying some kernel tuning to reduce jitter may help to eliminate the remaining accuracy issues. These issues are part of ongoing operating systems research [11].

5 Acknowledgements

This research was supported by the National Science Foundation (NSF) under grant number CNS-0833882. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF or Carnegie Mellon University.

References

[1] APPOSITE TECHNOLOGIES. Linktropy 4500. `http://www.apposite-tech.com/products/4500.html`.

[2] CARD, S. K., MORAN, T. P., AND NEWELL, A. *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Associates, Inc., Mahwah, NJ, USA, 1983.

[3] COMBS, G., ET AL. Wireshark. `http://www.wireshark.org/`.

[4] FABINI, J., REICH, P., AND POROPATICH, A. A generic approach to access network modeling for next generation network applications. *Networking and Services, 2008. ICNS 2008. Fourth International Conference on* (March 2008), 254–260.

[5] FOLKINSHTEYN, D. PyKeylogger. `http://pykeylogger.sourceforge.net/`.

[6] GALLETTA, D. F., HENRY, R. M., MCCOY, S., AND POLAK, P. When the Wait Isn't So Bad: The Interacting Effects of Website Delay, Familiarity, and Breadth. *Information Systems Research 17*, 1 (2006), 20–37.

[7] GRAY, W. D., SCHOELLES, M., KOTFILA, C., GAMARD, S., AND VEKSLER, V. D. Visualization-Interaction Architecture. `http://www.cogsci.rpi.edu/cogworks/?view=modules.research.spec&id=63`.

[8] KUKREJA, U., STEVENSON, W. E., AND RITTER, F. E. Rui—recording user input from interfaces under windows and mac os x. *Behavior Research Methods 38*, 4 (2006), 656–659.

[9] LIEUALLEN, A., BOODMAN, A., AND SUNDSTRÖM, J. Greasemonkey. https://addons.mozilla.org/en-US/firefox/addon/748.

[10] LINUX NETWORK DEVELOPERS. netem. http://www.linuxfoundation.org/en/Net:Netem.

[11] MANN, V. Operating system jitter. http://domino.research.ibm.com/comm/research_projects.nsf/pages/osjitter.index.html.

[12] MAX PLANCK INSTITUTE FOR PSYCHOLINGUISTICS. Elan. http://www.lat-mpi.eu/tools/elan/.

[13] SANDKLEF, H. GNU Xnee. http://www.gnu.org/software/xnee/.

[14] ZIMET, M. *Record Extension Protocol Specification*. Network Computing Devices, Inc., 1994.