An inaccurate-timing filter for reaction time measurement by JAVA applets implementing Internet-based experiments

JAN EICHSTAEDT

Universität der Bundeswehr Hamburg, Hamburg, Germany

JAVA applets are not accurate enough to measure reaction time unless precautions are taken. A response-filtering technique is introduced that identifies inaccurate measurements by testing whether the client-side JAVA implementation produces false measurements of a time interval of known length during the measured reaction time. Reaction times that indicate such inaccuracy are discarded to increase the reliability of the remaining data. Three studies provide illustrative data on the effectiveness of the new technique, using 11 different computer systems. Study 1 showed differences in the accuracy of different clients and higher variability of the obtained means with applets than with native programs. Study 2 showed sharply increased variability when additional loads were imposed on the clients during reaction time measurement. The response-filtering technique diminishes this variability. Study 3 indicated more accurate measurement, with the new technique being able to reveal even smaller differences in difficult technical conditions.

Wouldn't it be great? Early on Friday morning, you, a cognitive psychologist interested in reaction times, have a bright idea for a very telling experiment on task shift costs in which different kinds of components of a Web portal's user interface are used. Since the idea might really be insightful, you want to know the outcome immediately. So you sit down and modify a computer program of the experiment's prior version to incorporate your new idea. Because you implement it as a JAVA applet, it only takes you another half an hour to write down the short instruction in HTML and add an applet tag. Your experiment is on the Internet within an hour. Then you have breakfast and go off to teach. After a nice day in the Lecture Theater, your data collection is complete, your CGI script has logged the 48 participants' reaction times (RTs) and also performed a first statistical analysis. The deviations look good. The differences in average RT fit your hypotheses, and the effect is huge and, of course, highly significant. Have a nice weekend!

Unfortunately, this has been only a fairy tale until now, because of the inaccuracy of RT measured with applets. Your idea may still be a good one, and the differences between conditions are large. However it is much more probable that the times actually measured by the applet are so inaccurate that everything looks like a mess.

In general, Internet research is prospering (e.g., Janetzko, Hildebrandt, & Meyer, 1999; Klauer, Musch, & Naumer, 2000; Krantz, Ballard, & Scher, 1997; Lawson & Todd, 2000; Michalak, 1998; Reips, 1996, 2000; Welch & Krantz, 1996). Stevenson, Francis, and Kim (1999), as well as Malloy and Jensen (2000), have used JAVA to realize demonstration experiments for teaching. These are already applications of JAVA in the field of psychological experimentation that are useful because they do not depend on precise RT measurement. Moreover, there are quite a few studies that use JAVA applets on measures other than RT. For instance, Lawson and Todd investigated the role of similarities and differences in learning novel visual categories by measuring the participant's learning curve by counting trials until the learning criterion is reached.

To sum up, the technique for running experiments over the Internet is already in use, although not for accurate measurement of RTs. Among the most ambitious applications of the Internet as a medium for running experiments are implementations in JAVA. This programming language not only allows the setting up of complete programs—namely, applets—to be embedded in Web pages, but also does so with the same source code for all current computer systems. As a consequence, the applet becomes tightly integrated into the Web browser, because the applet object instantiates the appropriate objects of the browser (Gosling, Joy, Steele, & Bracha, 2000). For the participants of the study, handling is very convenient. They need merely to surf to the Web site hosting the experiment as a JAVA applet. Possibly, there is no other technique that better fulfills the needs of an experimenter on the Internet. However, the applet's lack of accuracy does not enable the experimenter to measure RTs.

I thank Jonathan Vaughan for his very helpful comments, which greatly improved the quality of this paper. I also thank Bettina Berendt, Norman Hendrich, Mike Hübner, Dietmar Janetzko, Ulf Reips, Claudia Rolko, Patrica Tegtmeier, and Julian Todd for their helpful comments. Correspondence concerning this article should be addressed to J. Eichstaedt, Universität der Bundeswehr Hamburg, Holstenhofweg 85, D-22039 Hamburg, Germany (e-mail: jan.eichstaedt@unibw-hamburg.de).

The Problem

JAVA is an object-oriented interpreter language. To run JAVA programs on all computers without modification, each computer system first needs an implementation of a so-called virtual machine (Gosling et al., 2000). The problem partly lies in the variety of client types—that is, the different virtual machines. Some virtual machines use the underlying operating system and processor in such a way that RT measurement turns out to be accurate, but other implementations fail to do so. Moreover, given multiple programs running simultaneously on one client (multitasking), one and the same computer sometimes is accurate enough and sometimes is not. So it might be possible that an implementation that is accurate under normal conditions becomes inaccurate when, for example, the mail client (running in addition to the Web browser and the virtual machine) begins to download an attachment. Even if no programs other than the browser and the virtual machine are running, browsers as well as virtual machines are multithreaded applications. They run processes or tasks in parallel. For example, the browser, while displaying the experiment program, may simultaneously request another page from elsewhere on the Internet. The processor may become overloaded if that second page contains elements that take up a lot of resources, like an animated GIF, another JAVA applet, a Quick Time movie, or the like. Unfortunately, the experimenter cannot control such activity. Applets that implement experiments on the Internet must therefore at least be able to prevent the impairment of the measures that is provoked by such “misbehavior.”

However, there is one problem that no effort in Java programming can account for. The resolution of the RT measurement done by a JAVA applet will never be better than the underlying operating system of the client allows. This is due to the interval with which the operating system looks for keystrokes or mouse clicks. Some do so only every 50 msec (Windows 95, 98); others have shorter intervals. The resolution of RTs taken by standard input events will never be better. This input resolution must be distinguished from the resolution of internal timing, which is much better. Some suspect the internal resolution of Macintosh systems to be only 1/60 sec, owing to the `TickCount()` function of the Event Manager (an operating system library). However, there is the new function `Microseconds()`, which has at least millisecond resolution. Furthermore, even before this new function was invented, it was already possible to gain millisecond resolution on the Macintosh by using the functions `IncsXTime()`, `PrimeTime()`, and `RmvTime()` (Apple Computer, 1992b; Peterson, 1998). According to Apple Computer (2000), the Macintosh system's internal timing resolution within the standard virtual machine (MRJ 2.2) is up to the millisecond, although in older systems (below G3) this is achieved with a workaround. However, a timing resolution of 1 msec does not guarantee a timing accuracy of 1 msec. All new browsers on the Macintosh use this virtual machine. On Windows systems, a lot of

different virtual machines are in use for which the internal timing resolutions are unknown but, in all probability, higher than the standard input retrieval. To sum up, the resolution problem is not Macintosh specific but stems merely from the interval with which the standard input events are retrieved. It prevents RT measurement done by a JAVA applet from gaining a resolution up to the millisecond. However, given the problem of inaccurate timing owing to multitasking, the resolution problem is minor. So the present paper will address only the inaccurate-timing problem.

There are only a few attempts to cope with the inaccurate-timing problem of JAVA-based reaction time measurement. One such JAVA-based experiment is that of Hecht, Oesker, Kaiser, Civelek, and Stecker (1999), who studied motion interpolation. Since this involved time-critical graphics animation, they were careful about the accuracy of the client systems. In order to estimate whether stimulus presentation and timing were accurate enough, they analyzed preexperimental calibration trials. That is, they checked whether the client system's performance was high enough. If the client system passed this test, they recorded RTs without checking further for false measurement. Unfortunately, this kind of procedure does not rule out the possibility of some new source of interference arising later in the experimental trials. An ongoing check is desirable, but as Hecht et al. rightfully pointed out, such checks consume valuable CPU time and therefore may interfere with time-critical processes. Nevertheless, Janetzkio et al. (1999; Hildebrandt, Meyer, & Janetzkio, 1998) took this approach. They were able to test each RT measurement, thus gauging whether it might be disturbed. This means that they were able to check even at later stages of the experiment. Yet this procedure, like that of Hecht et al., was based on testing the system's performance. The logic is the same as that in the approach of Hecht et al. Only powerful clients have enough computing capacity to measure the RT accurately enough. But this is a quite indirect approach. Rather, checking for performance in parallel in itself disturbs the RT measurement, the more rigorous the checking is.

The Inaccurate-Timing Filter's Approach

The inaccurate-timing filter identifies inaccurate RT measurements by testing whether the virtual machine makes false measurements of a time interval of known length. This known interval is a sleep interval of, let us say, 50 msec. In order to initiate the sleep interval, the *sleep(d)* method is issued on the control thread, which suspends it for the specified duration *d*. While a thread is suspended, it takes up virtually no processing power. Nevertheless, the virtual machine measures, by a specialized method, whether the duration *d* is exceeded. In addition to that, the inaccurate-timing filter also takes the time before and after a sleep interval of 50 msec (see Figure 1). However, in this case, it does so by a call to the method *System.currentTimeMillis()*, which is different from the method of the virtual machine that controls the dura-

```

start=System.currentTimeMillis();
    sleep(d);
end=System.currentTimeMillis();
    .
    .
    .
if (abs(end-start)>d+tolerance || startDelay>reactionTime*.625)
    reactionTimeAccuracy = false;

```

Figure 1. Simplified source code fragment for the detection of reaction time inaccuracy in JAVA applets.

tion d . If everything functions well, end time minus $start$ time necessarily must be 50 msec. It is possible that the system might not detect inaccuracy in the sleep duration if that duration were to be verified by using the same timing mechanism. In fact, in the end, all timing methods make use of the same transistors in the silicon of the processor. However, at least those parts of the object hierarchy of the applet within the virtual machine that measure time do differ. That is, the inaccurate-timing filter runs two different software clocks at the same time. If something interferes—for example, a simultaneous download of an attachment by the mail client—then one of the two software clocks will be more affected than the other. This in turn leads to inconsistencies in the measurement of the interval of known length, which the inaccurate-timing filter detects.

The implementation of the above principle is as follows. There is a main thread, which carries out the normal business of taking the RT by means of calling the *getWhen()* method on the keydown event object. This is supposed to be more precise than just calling *System.currentTimeMillis()*, because *getWhen()* records the time at which the *keydown* event happens to occur, rather than the moment at which the virtual machine returns to the *keydown* handler in the applet. Equally important, there is a second thread, which starts just before the RT measurement starts. This control thread runs in parallel and tests whether the RT measurement is correct.

This has something in common with Janetzko et al.'s (1999) approach to reducing errors in RT measurement. Janetzko et al. ran a parallel benchmark test that evaluated the system's capacity with a procedure that tends to result in overloading. The inaccurate-timing filter, on the other hand, tests the system's ability to measure time intervals directly, rather than drawing on the system's global performance. So only the measurement of the sleep interval is iterated again and again until the RT is recorded in the main thread. Consequently, the detected timing errors cumulate. As the control thread measures nothing other than sleep intervals, it is designed to take up as little processing power as possible, in order to avoid interference with the RT measurement of the main thread.

All the measurements of the sleep interval that the control thread identifies as false lead to the assumption that the measurement of the RT, which is actually done on the parallel main thread, is also wrong. So this par-

ticular RT is ruled out, thus preventing corruption of the RT's statistics. Although this is obviously desirable in the case of a real error in measurement, it is unfortunate if the control task alone fails while the main task remains unaffected. Such a situation would produce false alarms—that is, an RT is discarded although it was correct. Misses are also possible—namely, in the case in which there is a false RT in the main thread but a correct measurement of the sleep interval in the control thread. Hits and correct rejections are also produced. Of course, the inaccurate-timing filter is supposed to reject inaccuracies correctly.

Please note that the inaccurate-timing filter can never correct a false measurement. It can only detect it in most cases. This is because it cannot be distinguished whether the sleep interval has been distorted by some malfunction in the virtual machine or the measurement of the sleep interval is false. In either case, the control thread would produce numbers, which cannot be used to correct the measurement in the main thread.

There is a further diagnostic that is used to figure out whether the RT measurement has failed. One can assess the difference in time from the moment [recorded by a call to *System.currentTimeMillis()*] of issuing the *start()* method on the control thread object to the time [recorded by a further call to *System.currentTimeMillis()*] at which the control thread actually has started its run method. If the difference—that is, the delay until the control thread finally starts—exceeds 5/8 of the measured RT (a value based on experience), the system is likely subject to interference. This, as a second diagnostic, is taken into account in addition to the criterion detailed above. So if the start of the control thread is delayed or the two different software clocks within the control thread differ substantially, the RT measurement is considered to be false.

In order to tune the error check to the specifics of the different operating systems, the applet first assesses the type of the operating system via a call of the *System.getProperty()* method. Then it adjusts two parameters accordingly. First, the duration of the sleep interval must be adjusted, in order to control the processing power absorbed by the applet. In actual testing, it turned out that the Macintosh systems up to Version 9.04 needed a longer duration of the sleep interval (200 msec) than did the Windows systems (50 msec). Second, each operating system needs an adapted tolerance interval, in which the accu-

mulated differences between the actual and the measured sleep intervals may lie, without rejecting the parallel RT measurement on the main task as false. Actual testing revealed that values of 50 msec for Windows NT4 and NT5 (Windows 2000), 70 msec for Windows 95 or 98, and 90 msec for Macintosh fit best. However, the inaccurate-timing filter provides no quantifiable error term. Rather, exceeding the tolerance interval or a delayed start of the control thread are diagnostics that indicate only that interference might have distorted the RT measurement at all.

Turning attention to the implemented applet, it runs on all computer systems with full support of JDK 1.1 (Sun Microsystems' [1997] second release JAVA development kit, which is most common today). No additional permission by the client is required; it only uses methods and classes that are allowed, even at the highest security level. The inaccurate-timing filter's source code is available from me upon request.

To summarize, the approach is to give the virtual machine a chance to yield false time measurements in a situation in which you know the supposed outcome. If a false time measurement in the control thread is detected, the ongoing measurement of the RT in the main thread is assumed to be false as well and is filtered out. Thus, the error check becomes an inaccurate-timing filter.

How to Test the Inaccurate-Timing Filter

Generalizing from a sample of arbitrary hardware-software combinations to all the computers in the world is not possible; computers differ too greatly from each other. Nevertheless, even testing a small number of computers might result in the inaccurate-timing filter's failing, so it is worth doing. The value of such an evaluation will be improved if that small number of computers is diverse. To make it a worst case scenario, older computer systems will be used as well. Moreover, the applet will be run in an environment containing as much interference as possible—for example, by running a Quick Time movie while simultaneously measuring RT. That is, it will be evaluated whether the inaccurate-timing filter is capable of filtering out inaccurate RT measurements even in the most difficult technical conditions. This will make it more likely that better computers under better conditions will do at least as well as the ones evaluated.

In order to compare the accuracy of the applet-based measurement with that of a native program without the influence of JAVA, those programs must be developed for both of the two common operating systems—namely, MacOS and Windows. Unfortunately no single software solution for both of the two operating systems was available. However, the programming of these two programs need not be too sophisticated, because the applet's measurement will always be worse, as was explained above. So the assumed difference in standard deviation of a native program, as compared with an applet, should show up anyway.

To evaluate the recording of RT, an external input at a reliable and known rate is needed. Keyboards of IBM-

compatible computers have small circuits that determine, among other things, the keystroke repetition rate and hold the delay between two keystrokes quite constant. This was used by Myors (1999), who took it as the basis for testing a millisecond timer routine programmed to run on interrupt level on different PC operating systems. In the case of DOS in its most basic variant, he found the average keystroke delay's standard deviation to be as low as 0.0648 msec. So the keystroke repetition rate initiated by the keyboard chip's automatic repetition feature is accurate enough to produce a constant input with minimal variance.

In the case of Macintosh systems, the Event Manager (a part of the operating system) generates an auto-key event (Apple Computer, 1992a, pp. 2–38), which is interpreted by the virtual machine as a constant stream of keydown events. So in the case of IBM-compatible computers, the inaccurate-timing filter's accuracy is testable on a reliable basis of external origin. For Macintosh computers, the input comes from the operating system on which the virtual machine is running. That is, the input itself could be affected by the programs running simultaneously on the system, which ought to disturb the JAVA-based RT measurement. The following evaluation studies will show whether or not this operating-system-generated input is stable enough. However, in both cases, the input comes from outside the virtual machine. The delay is adjustable on all computers, allowing for comparisons of different delays. Although this seems to be a useful approach, it also has some limitations. The minimum repetition rate—that is, the longest delay between keystrokes—is not the same for different computer systems, and the disturbances that lead to false measurement are different, too. This prevents the study from comparing the different computer systems. Nevertheless, the inaccurate-timing filter should diminish disturbances and reveal differences in RTs in each of the tested systems that would otherwise be missed.

FIRST STUDY

Method

Overview. To lead off, a simple survey is needed to illustrate the assumed dissimilarities when measuring keystroke repetition with different kinds of computer systems in normal operation. More important, the aim of the study is to show the difference between JAVA and non-JAVA RT measurement. So, while running the applet, no additional applications were executed except the Web browser. In the statistical analysis, the standard deviation of the average keystroke repetition obtained by the native programs is compared with the standard deviation obtained by the applet, which implements the inaccurate-timing filter. The applet's standard deviations presumably will be higher. Moreover, the standard deviations of the applet, with and without the inaccurate-timing filter, are compared with each other, too. Even if something goes wrong with the measurement in normal operation, the error should be filtered out, thereby lowering the standard deviation substantially.

Apparatus. Eleven different computer-operating-system-browser combinations (see Table 1, first column) ran the same JAVA applet, implementing the inaccurate-timing filter, as was described above.

Table 1
Comparison of the Standard Deviations Obtained by Applet With and Without Inaccurate-Timing Filter and by Native Programs

Computer Type	JAVA Applet Without Inaccurate-Timing Filter			JAVA Applet With Inaccurate-Timing Filter			Native Program		<i>p</i>	<i>p</i>
	<i>M</i>	<i>SD</i>	<i>n</i>	<i>M</i>	<i>SD</i>	<i>n</i>	<i>M</i>	<i>SD</i>		
Mac 604-120,8.1,ie4.5	2407	15.44	30	2407	13.69	26	2394	0.50		.01
Mac 604e200,8.1,ie4.5	2430	132.64	30	2406	13.12	29	2394	0.81	.01	.01
PowerBook G3-400,9.04,ie4.5	2390	62.19	30	2402	9.75	29	2394	0.30	.01	.01
PC i486-33DX, w95, ie4.0	382	37.02	30	376	20.05	27	375	0.52	.01	.01
Toshiba P-100, w95, ie4.0	566	29.79	30	558	25.93	23	564	6.64		.01
Toshiba P-MMX 233, w95, ie4.0	563	24.26	30	557	24.26	30	565	4.35		.01
PC Celeron 233, NT4, ie5.0	403	4.27	30	403	4.27	30	402	4.97		
PC Celeron 233, NT4, n4.08	403	4.62	30	403	4.62	30	402	4.97		
PC PII-MMX 233, NT5, ie5.0	379	4.43	30	378	4.43	30	378	4.47		
PC PII-MMX 233, NT5, n4.08	379	3.90	30	379	3.90	30	378	4.37		
PC K6 266, w95, n4.7	515	27.63	30	508	24.71	25	516	3.84		.01

Note—Keystroke's means (in milliseconds) and standard deviations (*SDs*) obtained by the applet with and without the inaccurate-timing filter and, additionally, the means and *SDs* obtained by the native programs. The first column displays the type of computer system, which reads as follows: processor type, operating system, browser type. The last two columns give the significance of the difference in *SDs* with inaccurate-timing filter, as compared with those without a filter, and of the difference in *SDs* with inaccurate-timing filter, as compared with the *SDs* obtained by the native means, respectively.

The Windows systems were loaded only with the standard dll-files of the operating system. Similarly, the Macintosh systems were booted only with the necessary system extensions and control panels. File sharing was turned off. In the settings of the browser, the JIT compiler (just-in-time compiler, an option that makes the execution of applets faster but also has deep impact on how the applet is executed) was turned on.

The applet was embedded in an HTML document served from an HTTP server that was hosted on a SUN Ultra. This server additionally received the results transmitted by the applets from the client systems by means of a CGI script (see Kieley, 1996) programmed in PERL (see Schwartz, 1998).

In order to measure the keystroke repetition rate without the influence of JAVA, two native programs were used, one for the Macintosh systems and one for the Windows systems. Please note that neither is interrupt-oriented programming used nor is the inaccuracy imposed by the event or message loop of the operating systems circumvented. The event or message loop checks for new keystrokes or mouse clicks only every 50 msec in Windows 95 and 98 systems and about every 16–20 msec in NT and Macintosh systems. The Macintosh program used the *GetAKey()* function by Peterson (1998). For this study, the keystroke repetition rate was adjusted to a minimum, making the delay between keystrokes as large as possible. In Macintosh systems, this rate is adjustable in the keyboard control panel. In Windows systems, it is adjustable by using the keyboard panel in the system control folder.

Procedure. The client systems were switched on, the operating system booted, and the Web browser was started as the only application. Then the browser was steered in one step to the server and the HTML document with the applet. Next, the browser started the virtual machine and ran the applet, which provided a text field to get the testing conditions. After typing in processor type, operating system, browser type, and version, the user clicked the start button. The applet now waited for keystrokes that were given by pressing down the space bar once and then holding it down. After 30 automatic keystroke repetitions, initiated by the keyboard chip in Windows systems or by the operating system in Macintosh systems, the applet transmitted the results to the server by issuing a post request (see Harold, 1997, for details) to the server's corresponding CGI script and then stopped.

Results

As is shown in Table 1, the standard deviations of the keystroke repetitions were much higher when measured by the applet, as compared with a measurement by a na-

tive application programmed in C. On Windows systems, the means obtained by the applet, as compared with the means by a native application in C, were quite similar. The Macintosh systems appeared to show that the keystroke input generated by the MacOS is constant enough, allowing for substantial differences in standard deviations in two of the Macintosh systems (604e and G3). On some systems (604-120, 604e, G3, P-100, P-MMX, and K6), there is a small extra amount superimposed when using the Internet Explorer of Microsoft. (For the Macintosh systems, additional testing revealed that the results of the applet and the native program would be much more alike if the applet were run via the applet runner of Apple. But because the applet runner is not the usual tool of normal users, this alternative must be disregarded.) Because Netscape's implementation of the virtual machine up to Version 4.7 of the browser on Macintosh systems lacks full support for the abstract window toolkit (Gosling et al., 2000), which is needed for the error check, only Internet Explorer can be used on the Mac.

In three computer systems, the inaccurate-timing filter even reduces the standard deviation of keystroke repetition in normal operation substantially, whereas in two systems it does so only marginally. On the other hand, in six computers, the error check yielded nothing. So, in normal operation, the inaccurate-timing filter either successfully rules out false measurements or does nothing. In none of those cases did the inaccurate-timing filter corrupt the numbers.

SECOND STUDY

Method

Overview. In this study, it was investigated whether the inaccurate-timing filter would rule out interference stemming from running additional processes in parallel. The applet was running while calling additional Web pages or running further applications—for instance, a mail client—in parallel. Again, standard deviations of the average keystroke repetition with an inaccurate-timing filter should be lower than those without an inaccurate-timing filter.

Apparatus. The applet and the surrounding technique, as well as the tested computer systems, were the same as those in the first study.

Procedure. In addition to the operating procedure of the first study, this time considerable effort was spent on loading the computers with another task while simultaneously measuring the automatic keystrokes. This should simulate the difficult technical conditions that might stem from multitasking. Each system had its own vulnerabilities—that is, additional processes that could affect the measurement. The task chosen to be run in addition to and simultaneously with the inaccurate-timing filter was one of the following: Quick Time Movie, Real Player Movie, eMail client downloading an attachment, another Web page displaying an animated GIF, or another JAVA applet. In all instances, the worst cases were chosen by running the applet over and over again, each time yielding another 30 measurements. Only the worst five series were taken from each system for statistical analyses.

Results

Errors in measuring RT were increased when the Web browser or the computer system ran additional processes. However, as in Study 1, the different computer systems are hardly comparable, owing to the different maximum delays and, furthermore, the different kinds of imposed load. The Macintosh systems appeared to be more vulnerable than the Windows systems. In the case of the 604-120 and the 604e200 systems, both equipped with MacOS8.1 and Internet Explorer 4.5, just installing a large number of system extensions was already too much. For these two systems, nothing else but the browser with one window and the system extensions were running, which, nevertheless, resulted in huge interference with the measurement.

As can be seen from Table 2, the inaccurate-timing filter diminished the standard deviation of the RT drastically by ruling out false measurements in all but one computer system. However, the obtained standard deviations were still much higher than those obtained with native programs without load.

THIRD STUDY

Method

Overview. To test whether the inaccurate-timing filter helps to find differences in arithmetic means of RTs that otherwise would be missed, the study compares different keystroke repetition rates. The study compares two different delays between keystrokes that lie as near together as possible for each computer system. So this time, means are compared, using a *t* test for independent samples. It is expected that without an inaccurate-timing filter, the different keystroke repetition rates are more difficult to identify than with an inaccurate-timing filter. Certainly, the decline in standard deviations should show up again, when using the inaccurate-timing filter, as compared with not using it.

Apparatus. The applet and the surrounding techniques, as well as the computer systems, remained the same as those in the previous studies.

Procedure. Operation was the same except that, this time, only the three worst series of measurements were taken in two different conditions. In the first one, three series with a shorter delay between keystrokes were used, and in the second one, three series with a slightly longer delay between keystrokes were used. The different computer systems yielded different delays on equal adjustments. In all instances, the largest two adjustments lying closest together were chosen.

Results

Table 3 shows that without an inaccurate-timing filter, only six computers made distinguishable measurements of the two keystroke delays. Please note that even when a difference was detected, in the PPC604-120, i486-33DX, and K6-233 systems the underlying means were nevertheless still different from the more accurate means obtained by the native programs.

Detecting the difference was more reliable when the inaccurate-timing filter was used. However there is still an extra amount included in the means of the applet, as compared with the native program means. It appears that smaller means are more often subject to this error than

Table 2
Comparison of the Standard Deviations Obtained by Applet Under Heavy Load
With and Without Inaccurate-Timing Filter

Computer Type	JAVA Applet Without Inaccurate-Timing Filter			JAVA Applet With Inaccurate-Timing Filter			<i>p</i>
	<i>M</i>	<i>SD</i>	<i>n</i>	<i>M</i>	<i>SD</i>	<i>n</i>	
Mac 604-120, 8.1, ie4.5	2441	247.52	150	2418	16.39	24	.01
Mac 604e200, 8.1, ie4.5	2425	153.63	150	2409	13.34	142	.01
PowerBook G3-400, 9.04, ie4.5	2475	270.01	150	2401	35.75	65	.01
PC i486-33DX, w95, ie4.0	530	1098.76	150	367	66.71	16	.01
Toshiba P-100, w95, ie4.0	625	706.09	150	563	23.82	77	.01
Toshiba P-MMX 233, w95, ie4.0	603	151.98	150	561	37.04	125	.01
PC Celeron 233, NT4, ie5.0	440	247.14	150	403	61.01	117	.01
PC Celeron 233, NT4, n4.08	392	73.51	150	352	68.34	72	
PC PII-MMX 233, NT5, ie5.0	384	262.57	150	360	29.04	88	.01
PC PII-MMX 233, NT5, n4.08	391	178.21	150	382	27.78	97	.01
PC K6 266, w95, n4.7	3600	16421.90	150	509	36.66	103	.01

Note—Keystroke's means (in milliseconds) and standard deviations (*SDs*) obtained by the applet under the pressure of additive load with and without the inaccurate-timing filter. The last column gives the significance of the difference in *SDs* with the inaccurate-timing filter and without.

Table 3
Comparison of Means Obtained With a Long and a Slightly Shorter Delay Between Keystrokes

Computer Type	JAVA Applet Without Inaccurate-Timing Filter					JAVA Applet With Inaccurate-Timing Filter						
	Short Delay		Long Delay		<i>t</i>	Short Delay			Long Delay			<i>t</i>
	<i>M</i>	<i>SD</i>	<i>M</i>	<i>SD</i>		<i>M</i>	<i>SD</i>	<i>n</i>	<i>M</i>	<i>SD</i>	<i>n</i>	
604-120,8.1,ie4.5	247/133	252	710/199	682	−6.05*	159	30	62	210	14	36	−9.57*
604e200,8.1,ie4.5	227/133	211	273/199	206	−1.47	162	41	59	212	21	75	−9.12*
G3-400,9.04,ie4.5	291/133	183	371/199	372	−1.83	178	54	21	221	46	36	−3.20*
i486-33DX, w95, ie4.0	288/269	86	602/376	1405	−2.11*	311	58	12	373	73	10	−2.22*
P-100, w95, ie4.0	335/180	358	285/282	123	1.24	223	15	33	282	50	26	−6.41*
P-MMX 233, w95, ie4.0	243/217	134	322/282	153	−3.69*	218	16	72	284	35	68	−14.46*
Celeron 233, NT4, ie5.0	371/337	158	459/403	289	−2.53*	340	50	67	401	60	73	−6.54*
Celeron 233, NT4, n4.08	339/337	61	400/403	67	−6.44*	320	46	62	370	59	50	−5.11*
PII-MMX 233,NT5, ie5.0	318/269	78	411/376	334	−2.55*	309	9	49	369	21	56	−18.65*
PII-MMX 233,NT5, n4.08	367/269	401	408/376	223	−.83	320	27	34	381	18	59	−12.99*
K6 266, w95, n4.7	2,019/344	8150	2,897/516	14,882	−.49	342	27	73	509	46	53	−25.63*

Note—On the left side of the table, means (in milliseconds) and standard deviations (*SD*s) are obtained by the applet under the pressure of additive load without the inaccurate-timing filter. Here, the means obtained by the native programs are given after the slash. The last column of the left side of the table gives the *t* statistics of the means comparison. The right side of the table shows the means and *SD*s obtained by the applet under the pressure of additive load with the inaccurate-timing filter engaged. The last column displays the *t* statistic of the comparison of means obtained with the inaccurate-timing filter. **p* < .05.

are the larger means. Table 3 also gives the standard deviations, which are smaller when the inaccurate-timing filter is used. As before, the standard deviation reductions fall short, as compared with the deviations in native programs. As was hoped for, in all the computer systems, the inaccurate-timing filter enabled the applet to measure accurately enough to identify a small difference in keystroke repetition duration. On the other hand, this had a price, since the number of accepted measurements diminished greatly.

DISCUSSION

All in all, the inaccurate-timing filter might enable applets to be used as the basis for RT measurement experiments with an accuracy previously not met by applets. The inaccurate-timing filter can be used even on old computer systems (e.g., the 486-33DX system) and still allows measuring RTs. Moreover, it does not require anything else but a common Web browser; no certificates for security reasons or plug-ins for technical reasons are necessary.

Although this positive result is encouraging, it depends on the particular study—that is, the effect under investigation—whether the achieved accuracy is sufficient or not. Please note that differences in keystroke repetitions do not provide a good example with which real effects in the cognitive domain could be compared. This is due to the minimal variance of the keystroke differences' underlying process, which is of an electronic rather than a psychological origin. This makes it desirable to use a well-known cognitive effect, such as, for example, residual task shift costs (Allport, Styles, & Hsieh, 1994; Rogers & Monsell, 1995). Shift costs involve a difference in RTs of at least 43 msec, according to Rogers and Monsell's condition, which reduced the shift costs the most in their digit-letter switch paradigm. This would be an instance of a small but reliable effect that would pro-

vide the opportunity to test whether or not an applet on the Internet equipped with the new inaccurate-timing filter would replicate it.

Nevertheless, it must be kept in mind that an applet will never be more accurate than the underlying operating system of the client computer implementing the virtual machine. For example, some operating systems notify keyboard or mouse events only every 50 msec. So the measurement's resolution will never be better. However, given the heightened standard deviation found by the applet in keystroke repetition rates, which are almost perfectly constant, resolution as a matter is beyond the parameters of the data actually obtained.

Even though it is impossible to overcome the problem of imprecision of the underlying operating system, the inaccurate-response filter does its job well in all the tested situations and computer systems put to test. Older systems produced more inaccurate timings, which the filter removed. In the case of the 486-33DX system in Study 2, about 9/10 of the data were filtered out. If this is really due to the low performance of the system, it would be better to inform the participant before beginning to produce data that his or her computer is not appropriate for data acquisition. This could be achieved by testing the client's capabilities, as Hecht et al. (1999) did. However, in the case of the 486-33DX system, it was due to the strong interference of the simultaneously running programs (an eMail client downloading a big attachment, in this case). This can be seen in Study 1, where the same system, which ran without interference, had enough performance to measure RT with moderate accuracy, thereby filtering out only 1/15 of the data.

The studies showed that native programs are always more precise than applets. Thus, the inaccurate-timing filter should be used only when experimenting in a laboratory is not possible or, because of the research topic, is strongly undesirable. If the Internet is a must, redundant

experimental designs should be used, so that, in the case of interference, no missing values are produced.

In addition to the technical problems of measuring RT, such experiments have some other problems in common with Internet-based questionnaires or survey research. Schmidt (1997) enumerates incomplete responses and multiple submissions and proposes some solutions to these problems. Other problems he encounters may not apply to applet-based experiments on RT—namely, unacceptable responses, security, and data integrity. Nevertheless, the RT effect under investigation must be sharp and substantial and thus capable of overcoming the standard deviation, with which the applet is loaded anyway. The inaccurate-timing filter has shown the potential to ease this, at least to some extent, since there is a substantial reduction in standard deviation that the inaccurate-timing filter allows. Small differences in keystroke repetition have been found, which would have been missed without the inaccurate-timing filter. So if circumstances and the effect under investigation are suitable, the new inaccurate-timing filter adds further possibilities to Internet research, which otherwise remain unattainable.

REFERENCES

- ALLPORT, A., STYLES, E. A., & HSIEH, S. (1994). Shifting intentional set: Exploring the dynamic control of tasks. In C. Umiltà & M. Moscovitch (Eds.), *Attention and performance XV: Conscious and nonconscious information processing* (pp. 421-452). Cambridge, MA: MIT Press.
- APPLE COMPUTER (1992a). *Inside Macintosh: Macintosh toolbox essentials*. Reading, MA: Addison-Wesley.
- APPLE COMPUTER (1992b). *Inside Macintosh: Processes*. Reading, MA: Addison-Wesley.
- APPLE COMPUTER (2000). *Technical Q & A Java20: Using System.currentTimeMillis()* [On-line]. Available: <http://developer.apple.com/qa/java/java20.html>.
- GOSLING, J., JOY, B., STEELE, G., & BRACHA, G. (2000). *The Java language specification* (2nd ed.). Reading, MA: Addison-Wesley.
- HAROLD, E. R. (1997). *Java network programming*. Sebastopol, CA: O'Reilly.
- HECHT, H., OESKER, M., KAISER, A., CIVELEK, H., & STECKER, T. (1999). A perception experiment with time-critical graphics animation on the World-Wide Web. *Behavior Research Methods, Instruments, & Computers*, **31**, 439-445.
- HILDEBRANDT, M., MEYER, H., & JANETZKO, D. (1998, October). JAVA-basierte Versuchsdurchführung: Potentiale und Probleme am Beispiel von Reaktionszeitexperimenten [JAVA-based conducting of experiments: Potentials and problems shown by the example of reaction time experiments]. Paper presented at German Online Research (28.-29.10.), Mannheim.
- JANETZKO, D., HILDEBRANDT, M., & MEYER, H. (1999, March). JAVA-basierte Versuchsdurchführung im WWW: Potentiale und Probleme am Beispiel von Reaktionszeiten [JAVA-based experimenting in the WWW: Potentials and problems shown by the example of reaction times]. Abstract in E. Schröger, A. Mecklinger, & A. Widmann (Eds.), *Experimentelle Psychologie: 41. Tagung experimentell arbeitender Psychologen* (p. 15). Lengerich: Pabst.
- KIELEY, J. M. (1996). CGI scripts: Gateways to World-Wide Web power. *Behavior Research Methods, Instruments, & Computers*, **28**, 165-169.
- KLAUER, K. C., MUSCH, J., & NAUMER, B. (2000). On belief bias in syllogistic reasoning. *Psychological Review*, **107**, 852-884.
- KRANTZ, J. H., BALLARD, J., & SCHER, J. (1997). Comparing the results of laboratory and World-Wide Web samples on the determinants of female attractiveness. *Behavior Research Methods, Instruments, & Computers*, **29**, 264-269.
- LAWSON, R., & TODD, J. (2000, November). *The role of similarities and differences in learning novel visual categories*. Poster presented at the 41st Annual Meeting of the Psychonomic Society, New Orleans.
- MALLOY, T. E., & JENSEN, G. C. (2000, November). *Utah virtual lab: JAVA interactivity for teaching science and statistics online*. Paper presented at the 2000 Society for Computers in Psychology conference, New Orleans.
- MICHALAK, E. E. (1998). The use of the Internet as a research tool: The nature and characteristics of seasonal affective disorders (SAD) amongst a population of users. *Interacting With Computers*, **9**, 349-365.
- MYORS, B. (1999). Timing accuracy of PC programs running under Dos and Windows. *Behavior Research Methods, Instruments, & Computers*, **31**, 322-328.
- PETERSON, M. (1998). *Psychological toolbox: C/C++ functions for psychology experiments* [Manuscript and software]. Available: <http://or.psychology.dal.ca/~jc/PAPI.hqx>.
- REIPS, U.-D. (1996, October). *Experimenting in the World-Wide Web*. Paper presented at the 1996 Society for Computers in Psychology Conference, Chicago.
- REIPS, U.-D. (2000). The Web experiment method: Advantages, disadvantages, and solutions. In M. H. Birnbaum (Ed.), *Psychological experiments on the Internet* (pp. 89-114). San Diego, CA: Academic Press.
- ROGERS, R. D., & MONSELL, S. (1995). Costs of a predictable switch between simple cognitive tasks. *Journal of Experimental Psychology: General*, **125**, 207-231.
- SCHMIDT, W. C. (1997). World Wide Web survey research: Benefits, potential problems, and solutions. *Behavior Research Methods, Instruments, & Computers*, **29**, 274-279.
- SCHWARTZ, A. (1998). Tutorial: Perl, a psychologically efficient reformatting language. *Behavior Research Methods, Instruments, & Computers*, **30**, 605-609.
- STEVENSON, A. K., FRANCIS, G., & KIM, H. (1999). Java experiments for introductory cognitive psychology courses. *Behavior Research Methods, Instruments, & Computers*, **31**, 99-106.
- SUN MICROSYSTEMS (1997). *JAVA development kit version 1.1* [On-line]. Available: <http://java.sun.com/products/jdk/1.1/index.html>.
- WELCH, N., & KRANTZ, J. H. (1996). The World-Wide Web as a medium for psychoacoustical demonstrations and experiments: Experience and results. *Behavior Research Methods, Instruments, & Computers*, **28**, 192-196.

(Manuscript received December 1, 2000;
revision accepted for publication March 20, 2001.)