On the Latency of USB-Connected Input Devices

Raphael Wimmer
University of Regensburg
Germany
raphael.wimmer@ur.de

Andreas Schmid
University of Regensburg
Germany
andreas.schmid@ur.de

Florian Bockes
University of Regensburg
Germany
florian.bockes@ur.de

ABSTRACT

We propose a method for accurately and precisely measuring the intrinsic latency of input devices and document measurements for 36 keyboards, mice and gamepads connected via USB. Our research shows that devices differ not only in average latency, but also in the distribution of their latencies, and that forced polling at 1000 Hz decreases latency for some but not all devices. Existing practices - measuring end-to-end latency as a proxy of input latency and reporting only mean values and standard deviations - hide these characteristic latency distributions caused by device intrinsics and polling rates. A probabilistic model of input device latency demonstrates these issues and matches our measurements. Thus, our work offers guidance for researchers, engineers, and hobbyists who want to measure the latency of input devices or select devices with low latency.

CCS CONCEPTS

• **Human-centered computing** → **Interaction devices**; *Laboratory experiments*; • **Hardware** → *Communication hardware, interfaces and storage.*

KEYWORDS

latency, lag, USB, input, input devices, model

ACM Reference Format:
Raphael Wimmer, Andreas Schmid, and Florian Bockes. 2019. On the Latency of USB-Connected Input Devices. In *CHI Conference on Human Factors in Computing Systems Proceedings (CHI 2019), May 4–9, 2019, Glasgow, Scotland UK.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3290605.3300650

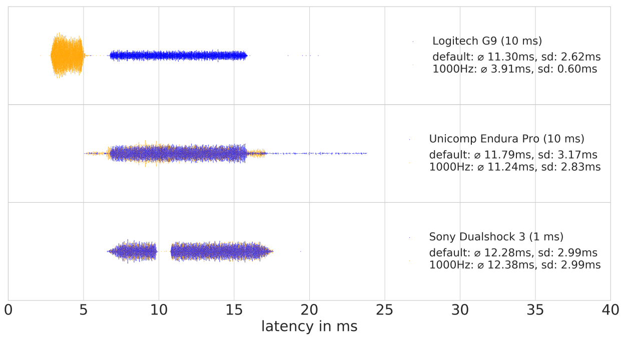

Figure 1: Latency of input devices varies significantly between devices. This becomes especially obvious when plotting the latency distributions gained from multiple measurements (blue). Reporting such latency distributions instead of average latency exposes individual quirks and common patterns (bottom). When forcing a 1000 Hz USB polling rate (orange), average latency and latency variance shrinks for most (top) but not all devices (middle).

1 INTRODUCTION

The amount of latency (or lag) in human-computer interfaces affects how effectively, efficiently, and satisfactorily users interact with a computer system. As shown by Ng et al. [21], users are able to detect the effects of latency in a user interface, down to an overall latency of 2 ms. In practice, a maximum latency of 20 ms for touch input and 2 ms for dragging actions is desirable [9]. High latency disproportionally increases task completion time for pointing tasks by slowing down feedback loops [18]. In certain computer games, such as first-person shooters or real-time strategy games, the amount of latency a user experiences decides over virtual life or death [12]. In experimental psychology, researchers conducting reaction time studies are concerned about the latency of computers tainting the results [24]. When controlling surgical robots or vehicles, spikes of high latency may cause critical situations and real physical harm [25]. Therefore, reducing and controlling latency is of great importance for designers of such systems. Attig et al. give a good overview of recent findings on the effect of latency on human performance [2].

In order to characterize or reduce the overall latency of a system, it is necessary to isolate and measure individual sources of latency, such as input devices, output devices, network connections, or applications. In recent years, several researchers and hobbyists have started measuring the latency of input devices. However, we have found that previous research reports imprecise or incorrect latencies,

CHI 2019, May 4–9, 2019, Glasgow, Scotland UK

incorrectly uses overall latency as a proxy for input latency, and neglects the important effects of polling rate and phase on latency.

We present a simple method for precisely and accurately characterizing the latency of common input devices, such as computer mice, keyboards, joysticks and gamepads. To achieve this, we explicitly choose three limitations:

Focusing only on USB-connected input devices makes it easier to compare devices and reason about technical issues. While sensors may be connected to a computer through various interfaces (e.g., I2C, SPI, UART), the Universal Serial Bus (USB) [27] is by far most common wired peripheral interface on desktop computers, notebooks, and mobile phones at the moment. The wide choice of available input devices allows users and designers to actually choose an input device based on its latency.

Limiting latency measurements to binary button presses allows for more accurately characterizing input latency. Unlike mouse or joystick movement, button presses are easy to reproduce repeatedly, have a quite clearly defined starting point (when electrical contact is made), and have no inherent noise. Furthermore, mice, keyboard, and gamepads all contain buttons, allowing to compare their latency.

Ignoring the mechanical latency of a button press improves accuracy, precision and reproducibility of the measurements. The amount of time between a user pressing a button and it closing an electrical contact is undoubtedly an important characteristic of an input device. However, as Oulasvirta et al [22] show, pressing a button is an inherently complex neuromechanical process. Thus, it is conceptually hard to define at which point in time a button press is initiated. Furthermore, humans are quite bad at precisely repeating the same movement thousands of times.

For conducting the measurements, we implemented *LagBox*, a simple Raspberry-Pi-based tool that measures latency of USB-connected input devices by repeatedly electrically triggering a button on the device and detecting the input event on the Raspberry Pi. LagBox can automatically conduct thousands of latency measurements per input device, reproducibly collecting more data in a shorter time period and with higher accuracy than manual approaches. Our measurements show significant differences between devices and highlight some peculiarities and commonalities (Figure 1). Most importantly, we found that latency distributions differ so much between devices that only reporting average latency and standard deviation may be misleading. In order to better understand and explain how these latency distributions are generated, we developed a probabilistic simulation of input latency that allows reproducing many of the latency distributions we observed.

Our work is intended to provide guidance and tools for researchers, practitioners, and enthusiasts who want to measure or improve latency. The main contributions in this paper are: a model of latency in input devices, a set of guidelines for measuring and reporting latency, a robust method for measuring latency of USB-connected input devices with high accuracy and precision, and a collection of measurements illustrating important characteristics of input device latency.

This paper is organized as follows: In the next section, we shortly introduce basic concepts of *Latency in Interactive Systems*. Afterwards, we give an overview of *Related Work*, present *A Model of USB Input Latency*, and discuss challenges when *Measuring and Reporting Input Latency*. We describe the implementation of our measuring system, *LagBox*, and present *Latency Measurements*, for 36 USB input devices gathered with LagBox. After a *Discussion* of limitations and general insights that can be gained from our measurements, the paper ends with a *Conclusion and Outlook* on future work.

2 LATENCY IN INTERACTIVE SYSTEMS

As famously described by the Card et al.'s *human processor* model [4], interactive systems are part of a continuous feedback loop wherein the user performs an input action which is received and processed by the system. The system's output is in turn perceived by the user who reacts to it. Latency in an interactive system slows down these feedback loops and thus increases the time needed to perform a task. The most important latency metric for interactive systems is *end-to-end latency*, also sometimes called *system latency* or *overall latency*. It usually refers to the delay between user input and perceivable system output[1]. We define end-to-end latency as the amount of time that passes between the user starting to change a physical state which is monitored by the system (e.g., pressing a button on a keyboard) and the system starting to output something in response to the user's action (e.g., as a change of screen content on a monitor, auditory, or haptic feedback). In order to reduce end-to-end latency, its individual components need to be analyzed and reduced. These *partial latencies* can be roughly grouped into *input latency*, *processing latency*, and *output latency*. We define *input latency* as the delay between the beginning of a physical state change (e.g., button press) and an application receiving notice of this state change. *Processing latency* is the delay between an application receiving notice of a physical or digital state change and the application handing over the results of the state change to the operating system (or

[1]A detailed description of the many individual steps involved when typing a search term into a web form and pressing *Enter* can be found at https://github.com/alex/what-happens-when

computer if no OS is involved). Processing latency may also typically include network latency, disk latency, etc. *Output latency* describes the delay between the output data being handed over to the operating system and it actually being presented to the user.
This rather crude grouping has the advantage that it mirrors the typical separation of concerns in developing, using, and studying interactive systems. Rarely do application developers also implement input/output hardware, corresponding operation system drivers, or input/output frameworks. As most interactive systems allow for adding or exchanging individual input or output devices, users interested in reducing the end-to-end latency of their system can choose appropriate input and output devices independently from the applications. As we discuss later in this paper, measuring just end-to-end latency does not allow inferring the actual latency of a component. Only by focusing on individual parts, we are able to identify *why* and *how* latency varies. In the remainder of this paper we focus on measuring input latency.

3 RELATED WORK

Measuring input latency has been a pastime of researchers, engineers, and hobbyists for decades. The latency measurement methods employed in previous research can be categorized in the following ways:

- directly measuring *partial latency* vs. using *end-to-end latency* as a proxy metric
- *destructive* vs. *non-destructive* rigging of input devices for latency measurements
- reporting *absolute latency* of a device vs. reporting *relative latency* compared to other devices[2]

3.1 End-to-End latency on Desktop Computers

Many previously presented approaches measure end-to-end latency, i.e. the time difference between an input event (e.g., the user pressing a button) and an output event (e.g., a change of screen contents). By varying only one part of the processing pipeline - such as the input device that is being used - one may indirectly determine latency differences caused by these changes und thus non-destructively measure relative, partial latency. This approach is often done using a video camera [10]. Industrial cameras and high-end mobile phones offer frame rates of several hundred or thousand frames per second, allowing for measuring latency with high precision. However, accuracy of such approaches is problematic. Video footage needs to be manually analyzed in order to find the point in time where an input event occurs. In many cases, it is not easy to precisely define the beginning of an input action. Casiez et al. [5] - who also give a good overview of related work on latency - present a non-destructive approach for measuring absolute end-to-end latency of a computing system. To this end, they place a computer mouse on a display that shows a moving pattern which stimulates the mouse into reporting movement. Measuring the time between pattern change and mouse event gives an estimate of end-to-end latency. The authors report results of measurements with different graphical toolkits and for different computer mice. In a later publication, Casiez et al. [6] present another method for measuring end-to-end latency. They attach a piezoelectric vibration sensor to the user's finger to capture tapping and button-pressing. An application running on a computer changes screen contents when receiving an button-press event. A photodiode is used for capturing this screen response. An Arduino board captures input from both sensors and measures the time between both events. While the authors' focus is on presenting a measuring method, they mention that output latency makes up most of the end-to-end latency they measured[3]. Casiez et al. report an input latency of 12 ms for a computer mouse polled at 125 Hz and 2 ms for a gaming mouse polled at 1000 Hz.

3.2 End-to-End Latency on Touch Screens

Deber et al. present a similar tool for measuring the end-to-end latency of devices with touch screens [8]. Their "Latency Hammer" uses a metallic probe that is placed on a capacitive touch screen. When connecting the probe electrode to ground, a touch event is simulated. A photodiode connected to the "Latency Hammer" captures the screen response.
Mooney and Koudritsky, engineers at Google, have published tools for measuring input latency of touch screens - Quickstep[4] and its successor WALT[5]. Both employ simple custom hardware which senses input events in the physical domain. The difference between the timestamps of the physical input event and of the digital input event (as recorded by the operating system's kernel) gives the latency of the touch screen. Using these tools, Mooney and Koudritsky found that typical tap latencies for touchscreens in mobile devices are in the range of 10 - 45 ms[6].

[2]i.e., the difference in latency between an input device and either a reference device (whose absolute latency does not need to be known) or multiple other devices.

[3]Leo Bodnar Electronics sells a *Video Signal Input Lag Tester* which allows for measuring the latency of TVs and computer monitors. http://www.leobodnar.com/shop/?main_page=product_info&products_id=212

[4]https://docs.google.com/document/d/1B3GfZkwAfHlQE5kAuHcn-wq8I - 7D5UF2Ap7C_CYk5co/

[5]https://github.com/google/walt

[6]Mark Koudritsky, personal communication, 2016-04-12.

3.3 Input Latency

While the aforementioned research allows for indirectly measuring the latency of input devices, few researchers have directly measured input latency. Most notably, Plant measured the input latency of eight computer mice, a keyboard, and a *response box*[7] connected to a computer via the serial port, PS/2 or USB [24] . An oscilloscope was used to measure the delay between a (simulated) button press on the device and a signal being emitted to the computer. Each measurement was repeated 80 times. Latencies recorded ranged from 6.5 ms to 60 ms for the mice. The keyboard had a latency of 28 ms, whereas the response box had a latency of only 1 ms. Plant et al. also report on previous attempts to measure mouse latency ranging back to 1990. Plant has published several other papers on this topic over the last decade, urging psychologists to measure and control input latency in their experiments [23].

Luu conducted experiments measuring the latency of multiple keyboards by manually triggering keys and inspecting the output of a logic analyzer attached to the USB cable [16]. The measurements are rounded to 5 ms increments and Luu estimates an error in the 1 ms to 10 ms range. Luu also reports on further experiments and sources concerning general latency of computers and terminals [15,17].

In parallel to academic research, computer gaming enthusiasts have explored ways to to measure the latency of input devices. Overclock.net forum member *uaokkkkkkkk* has compiled a comprehensive overview of these approaches[8]

A very simple, non-destructive way to measure the difference in latency between two computer mice is to connect both to the same computer and simultaneously press buttons on both mice. This is usually done by placing the left mouse button of one mouse on the right mouse button of the other mouse and rapidly pressing both together. Input device manufacturer *Bloody* offers a Windows application for this task[9]. The same approach also works for joysticks and keyboards. While simple and non-destructive, this approach requires a second mouse to be available and does not measure absolute latency. Furthermore, if the spring of the micro-switch in the first mouse is stronger than the spring in the second mouse, the second mouse button will always be depressed earlier, resulting in a systematic measurement error. In addition, the approach requires repeated manual action. This introduces additional random and systematic measurement errors depending on the user conducting the experiment.

Since 2013 the Japanese blog *rafalog*[10] has measured the relative latency of more than 50 mice by soldering a single push-button to the micro-switches of both a reference mouse (Logitech G300) and a mouse to be tested. Pressing this button simultaneously closes the electrical circuits of both mice, mitigating the major limitation of the aforementioned *Bloody* approach. The *rafalog* approach is destructive and only reports relative latency. However, as they use a Logitech G300 as reference mouse in all tests, whose absolute latency we report in this paper, absolute latencies for the other mice can be easily calculated. A commenter on *rafalog*, Chou Star, seems to have implemented a system that is quite similar to our LagBox[11]. They use the GPIO pin on a Raspberry Pi to toggle the electrical circuit of a mouse button while the mouse is connected to the Raspberry Pi. A video shows latency values for the G300 which are in the 3 - 5 ms range, roughly similar to the ones we measured. Another blogger, *sousu*, conducted absolute, end-to-end latency measurements for over 50 mice with another destructive method[12]. They soldered a phototransistor to the contacts of the mouse button and glued it to a computer screen. The measuring software turned the screen white causing the phototransistor to become conductive and trigger the mouse button. The latency between screen update and mouse event was measured. Latency values seem to be a few milliseconds higher than our measurements as end-to-end latency was measured.

Overall, we found that only few researchers and hobbyists investigated the latency of input devices so far. Most existing approaches either use end-to-end latency as a proxy for input latency or measure only relative latency. In general, only average latency, and sometimes standard deviation, is reported. As discussed in the following sections, both practices distort the actual latency profiles of input devices.

4 A MODEL OF USB INPUT LATENCY

When conducting preliminary latency measurements with our system, we were surprised by the latency distributions some devices exhibited (Figure 1). In order to better understand the observed data, we developed a simple model that helps to explain how complex latency distributions may arise from a combination of individual latency sources.

[7]A *response box* is an external computer peripheral used for reaction time experiments. It usually contains multiple buttons and/or lights and allows for low-latency, high-accuracy/-precision time measurements.

[8]http://www.overclock.net/t/1572872/firmware-added-button-delay-testing -attempts-tapping-mouse-buttons-lightly (most information is hidden behind the "spoilers" links)

[9]KeyResponse PK, http://www.bloody.tw/en/download.php

[10]http://web.archive.org/web/20160224102835/https://utmalesoldiers.blogspot.de/2013/02/114.html

[11]https://www.youtube.com/watch?v=u9z62MKX_Wg

[12]http://sousuch.web.fc2.com/DIY/mouse_latency/index.html

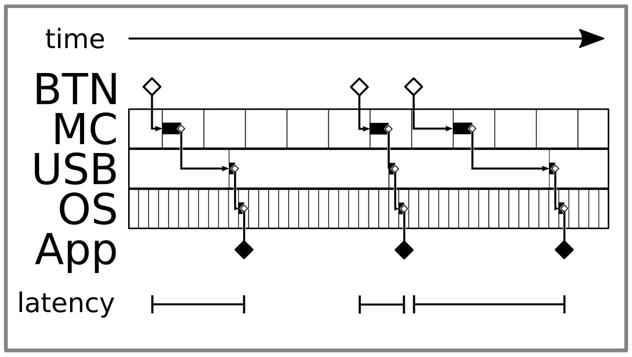

Figure 2: Simplified illustration of the path an input takes from button press to the input event arriving at the application. Each step adds constant latency (black bar). The microcontroller (MC), USB, and operating system (OS) also add varying amounts of latency depending when an event is handed over within a scanning/polling interval. Overall input latency therefore varies significantly depending on when the button is pressed.

4.1 Types of Latency in a Button Press

When a user presses a button on an USB-connected input device, the following individual steps happen, each with its associated latency (approximate orders of magnitude in parentheses):

1. the user first touches the input device,
2. the user overcomes activation force and triggers a mechanical switch (~ 20 ms [11]),
3. the mechanical switch closes an electrical circuit,
4. the closed circuit is detected by the device's controller chip (~ 1-20 ms),
5. after processing the sensor data the chip puts data into a USB buffer (~ 1-20 ms),
6. the host computer queries the USB device for new data (~ 1-10 ms),
7. the device sends data over the wire (0.001 ms),
8. the host computer notifies the OS about new data from the USB (0.001 ms),
9. the OS has processed the data and made it available to userland libraries (0.01 ms),
10. user code has received an input event from a userland library (0.01 ms).

The overall latency of a button press is therefore in the range between 23 and 70 ms, depending on user, button, and context. In this paper, we focus on non-mechanical latency comprised by steps 4 - 10. While each individual step contributes latency, the amounts and distributions of these partial latencies are very heterogeneous and device-specific.
Figure 2 shows a slightly simplified example of the involved latencies.

Detecting a closed switch (step 4) may be either done via a hardware interrupt in the controller or by regularly scanning the state of the switch. In general, hardware interrupts are consistently processed within microseconds, adding **constant latency**. Scanning introduces **variable latency**: if the button is pressed right before the controller checks its state, latency is low; if the button is pressed immediately after the controller checked its state, the controller will only find out about this the next time it scans the state of the switch. In most cases, hardware interrupts provide little benefit and add complexity to the code. As far as we know, most USB input devices sequentially scan switch state in the main loop of the controller code[13].
Processing the input in the device's controller (step 5) may take constant time in theory. In practice, different code paths may be taken which introduce a **set of distinct constant latencies**. For example, debouncing[14] or integrating the state of other sensors may require a variable amount of time.
These three types of latency - constant latency, variable latency, and a set of distinct latencies may be used to characterize each processing step. Of course, in reality constant latency is not perfectly constant but shows some spread due to physical noise. Variable latency may exhibit different distributions, most commonly uniform or normal distributions. While processing of USB events on the host computer (steps 8-10) usually occurs within microseconds, USB polling (steps 6-7) introduces variable latency up to several dozen milliseconds and thus warrants closer inspection.

4.2 The Role of the USB Polling Rate

HID-class USB input devices [28], such as keyboards, mice, joysticks, and gamepads, communicate with a host computer using *interrupt transfers*. Despite their name, interrupt transfers do not actually rely on hardware interrupts. Instead, the host computer periodically polls the USB device for new data or sends new data to the device [27]. In contrast to other transfer types, interrupt transfers are guaranteed a maximum upper latency limit. A device endpoint may specify a polling interval via its *bInterval* field which is transmitted during registration of the device with the host. This polling interval specifies how much time may pass *at maximum* between two polling events.

[13]If an input devices has many keys, such as a keyboard, matrix scanning is employed. This implementation choice does not substantially affect latency, however.
[14]The metal contacts in mechanical micro-switches tend to bounce slightly after impact, resulting in multiple short interruptions of the contact. The controller firmware has to filter out these 'echoes', e.g. by not accepting repeated button presses for a limited time. Some device manufacturers employ optical switches which do not require debouncing (e.g., http://www.bloody.tw/en/Products.php?pid=28, http://www.wooting.nl/)

According to the USB HID specification, for *high-speed* input devices (connected via USB 2.0) the polling interval may be in the range of 1 to 255 ms [28:69]. For *low-speed* input devices (USB 1.1), the minimum polling interval is 10 ms [27:51]. The actual polling interval may be shorter but is guaranteed to be not longer. On Windows, the maximum polling interval is limited to 32 ms [19]. In practice, polling intervals are rounded down to the next lower power of two, at least on Windows [19] and Linux [14]. This allows for better allocation of available bandwidth. Therefore, an USB HID device specifying a desired polling interval of 10 ms is usually polled every 8 ms, i.e. with a polling rate of 125 Hz. If the polled device does not have any new data available in its USB buffer, it sends a NAK packet. It is therefore possible and specification-conforming to poll a HID device at a higher rate than specified. For example, the *usb_hid* driver in the Linux kernel allows specifying a custom polling rate for HID mice and joysticks. Similar functionality is offered by vendor-provided and inoffical Windows tools. It is primarily used by gamers to reduce input latency. We investigated the effects of polling devices with the maximum rate of 1000 Hz. Results are described in the *Latency Measurements* section.

An important aspect of USB devices being polled at fixed intervals is that their latency in no small part depends on the polling rate. Even if an input device rapidly detects and processes a button press, the host computer only learns about the button press when it polls the device the next time. However, a low polling rate results not only in higher overall latency but also in significant variance in latency. This issue is rarely acknowledged at all in studies of input latency. Another important aspect to consider is *polling phase* of each latency component, i.e. the relative displacement between the scanning/polling intervals of devices within a processing chain. Overall input latency may vary a lot depending on the phase difference between the involved polling intervals. We describe both effects in more detail in the following section.

4.3 Simulating Input Device Latency

In order to verify whether our understanding of input device latency is correct, we developed a simple programmatic model of input latency for USB devices. As most device manufacturers do not publish the source code for their firmware, input devices are essentially black boxes. However, based on the assumptions described above and example source code providedme.g., by Cypress Semiconductors[15], one may build a model of a *typical* input device. Such a model has three purposes. First, the model allows us to validate our understanding of input device latency by comparing predicted latencies with empirically measured latencies. Second, the model allows for predicting latency distributions of input devices if components are known. Third, the model allows to find the latency properties of input devices by adjusting model parameters, such as keyboard scan rate, until the predicted latencies match measured latencies. In this paper, we focus on the first use case, showing that a simple combination of partial latencies may produce complex latency distributions.

[15]http://www.cypress.com/file/103181/download

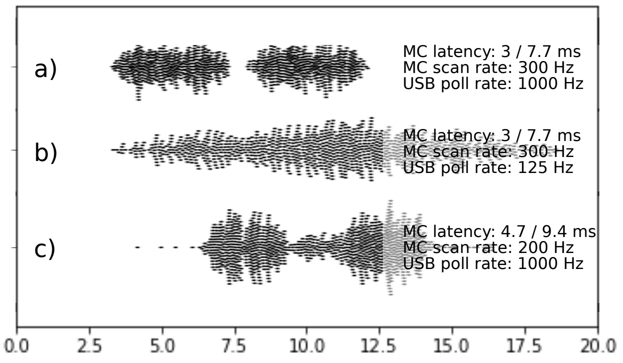

Figure 3: Simulations of input latency for a controller with two code paths. Small changes in partial latencies, scanning rate, polling rate, and polling phase produce distinctly different latency distributions.

In our model, each major latency component - button, microcontroller, USB bus, and OS - is implemented as a Python object that adds intrinsic latency and polling-related latency. The intrinsic latency can be either a constant value or randomly chosen from a set of constant latencies (which may exhibit a certain distribution as described above). Furthermore, each component has a polling/scanning rate and a polling/scanning phase. From these values, the polling-related latency is determined. When running the simulation, a randomized start time is passed to the *button* object which adds its intrinsic and polling-related latency and returns a new timestamp. This timestamp is in turn passed along by the *microcontroller*, *USB*, and *OS* objects which each add their own intrinsic and polling-related latency.

Figure 3 shows the latency distribution for 2000 simulated measurements. In this instance, the microcontroller had two "code paths" with different latency from which one was chosen at random. While the distinct latency of each code path is clearly visible in Figure 3a - where the USB polling rate was set to 1000 Hz / 1 ms, the latency distribution for polling at 125 Hz / 8 ms (Figure 3b) hides the effect of different code paths on latency. Figure 3c shows the simulated latency distribution for a controller with slightly different values for latency and scanning rate.

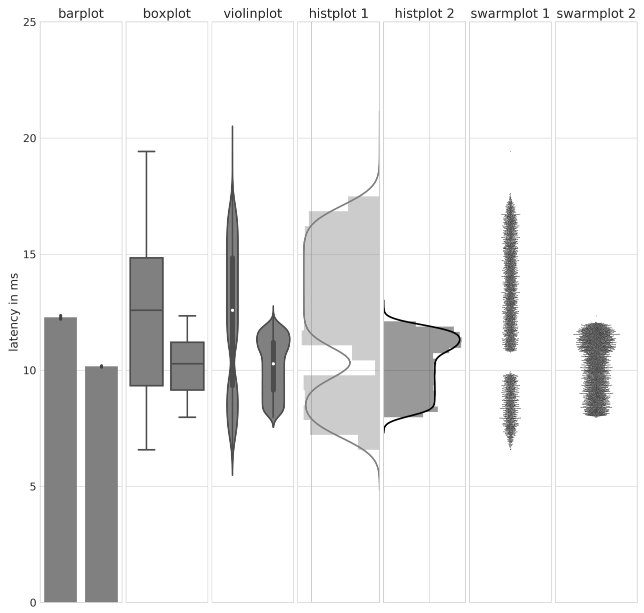

Figure 4: Actual latency measurements for two gamepads visualized in five different ways: a) bar charts, b) box plots, c) violin plots, d) histograms with kernel density estimates and rug plots, e) swarmplots. Only swarm plots and histograms faithfully show peculiar latency characteristics.

Adding many partial latencies which have a uniform or normal distribution always leads to a more or less normal-distributed overall latency if the ranges are in the same order of magnitude. However, partial latencies that have a distribution with much higher average or variance than the other latency distributions, dominate in the overall latency distribution. Thus, a low USB polling rate smoothes and broadens the overall latency distribution.

For reporting latency distributions, we strongly suggest using swarmplots which are similar to histograms but show each individual measurement as a dot. Other forms of visualizations, such as bar graphs, boxplots, violin plots, or histograms may hide important characteristics (Figure 4).

5 LAGBOX

In order to accurately and precisely measure input latency, we used LagBox [3], a system for automatically triggering the buttons of input devices and measuring how much time passes until the corresponding input event is received. This device was used for conducting all measurements presented in this paper. In order to simplify device design and make the device affordable for a wide audience, LagBox is built around a Raspberry Pi 2. A shown in Figure 5, an optocoupler is connected to a GPIO pin on the Raspberry Pi. The

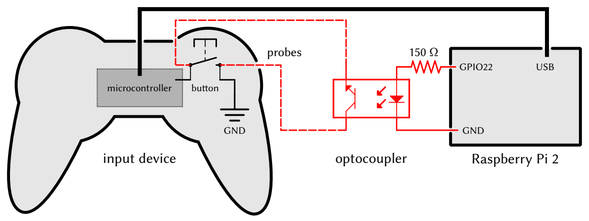

Figure 5: The LagBox measurement circuit. Via an optocoupler (Fairchild Semiconductors H11A817) connected to a GPIO pin, the Raspberry Pi triggers the button of an input device and measures the time until the USB event arrives.

switched side of the optocoupler is connected to the contacts of a button in the device that is to be tested. The connection can either be made via soldering or via micro-clips or pogo pins[16]. The device under test is also connected to the Raspberry Pi via USB. When the GPIO pin is pulled high, the optocoupler electrically connects both contacts of the button, thereby simulating a button press. This is registered by the input device's microcontroller which stores the new value in its USB buffer and transmits it to the host computer the next time the USB endpoint is polled.

LagBox offers two user interfaces - a versatile command line interface and a graphical frontend written in Python. A small application written in C handles measurements and logging. It is optimized for low overhead and low latency in the critical code path. The general testing algorithm is as follows:

```
start:
wait pseudo-random amount of time
set GPIO pin to activate optocoupler
record timestamp
loop:
  read from Linux device
  if new event of correct type available:
    record timestamp
    calculate and store latency
    goto start
  else:
    sleep for 10 µs
    goto loop
```

Potential latency sources in our measurement setup are the main loop of our measurement software, triggering the GPIO pin, and handling the USB packet. We have verified that all of these inherent latencies are in the low microsecond range, adding up to much less than 0.1 ms. Thus, we are confident that our setup accurately measures latency in the millisecond range with sub-millisecond accuracy.

[16]We used the Hirschmann MICRO-KLEPS clips and MPS2/0.64FT probes.

Between each two latency measurements, our code waits for a pseudo-random amount of time between 100 and 1000 ms in order to emulate button presses at random times. If one would not wait and trigger the button again directly after an USB event has been received, the next event would usually arrive exactly *polling interval* milliseconds later at the next time when the device is polled.

6 LATENCY MEASUREMENTS

In the following, we describe measurements that we conducted using LagBox. Overall, we measured the latency of 36 commercially available USB devices (including wired and wireless variants): 11 keyboards, 16 mice, and 9 gamepads. Our choice of devices was determined by three factors: availability, published latency measurements in related work, and popularity. Input devices which only had a PS/2 connector were measured using a PS/2-USB adapter reporting a *bInterval* of 10 ms. For each device and each choice of polling rate, 5000 latency measurements were conducted. After each 1000 measurements, we disconnected and reconnected the input device in order to mitigate any synchronization effects.

6.1 General Comparison

As shown in Figures 6 - 8, significant differences exist between devices and between device classes. Latency distributions are shown as overlayed swarmplots - blue for the default polling interval, orange for a forced 1000 Hz polling rate (see below).

6.2 Forced 1000 Hz polling.

While USB devices report the desired polling rate in their endpoint descriptors, the operating system may poll the devices more often - up to every millisecond (1000 Hz). The Linux `usbhid` driver offers a configuration option to manually define the polling interval in milliseconds for mice (`usbhid.mousepoll=N`), joysticks (`usbhid.jspoll=N`), and keyboards (`usbhid.kbpoll=N`)[17].
As only the *mousepoll* parameter was implemented when we started our research, we modified the usbhid driver so that the manually defined polling interval applied to all devices. For all tested input devices, we measured the latency once when polled at their requested polling rate, and once when explicitly polled at 1000 Hz. In Figures 6 - 8, latency distributions for the default polling rates are plotted in blue, whereas distributions for polling at 1000 Hz are plotted in orange. If the blue plot completely occludes the orange plot, this means that polling at 1000 Hz makes no difference – usually because the device is already polled at 1000 Hz by default.

[17]https://patchwork.kernel.org/patch/10299667/

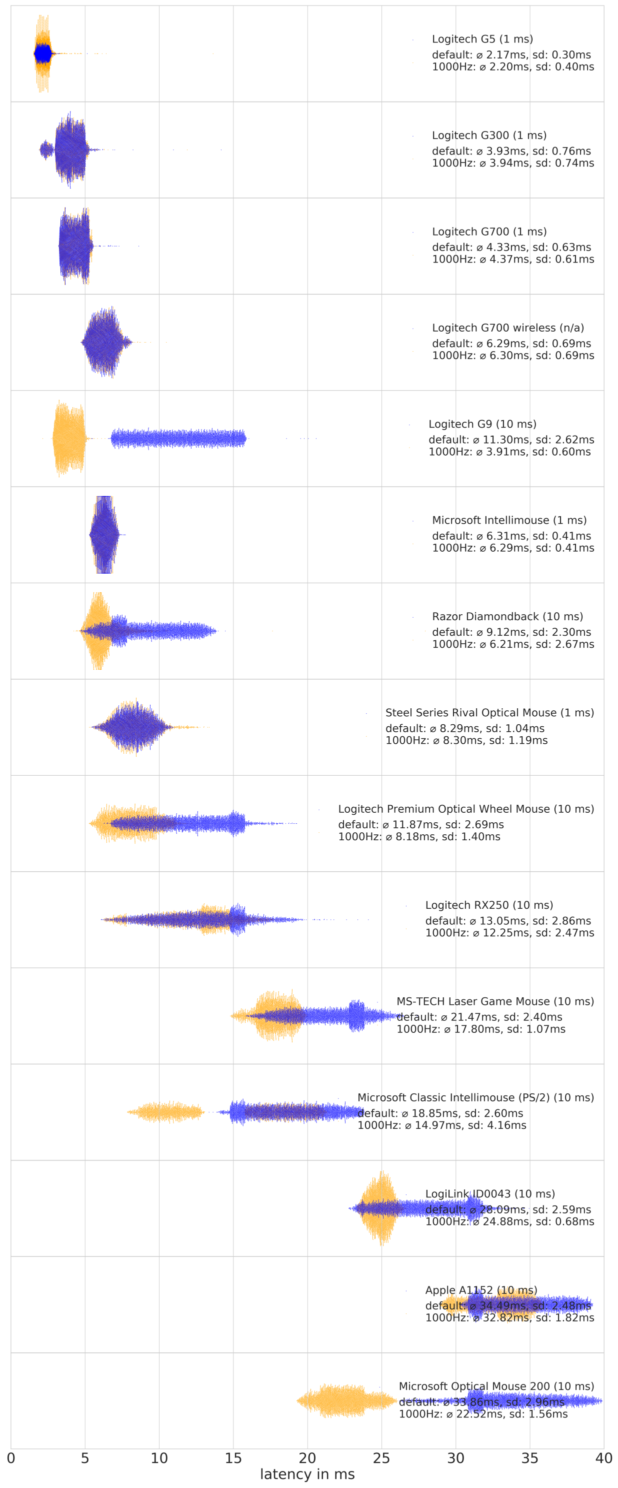

Figure 6: Latency distributions for USB mice. Default polling interval is given in the legend. Blue: default polling rate, orange: forced 1000 Hz polling rate.

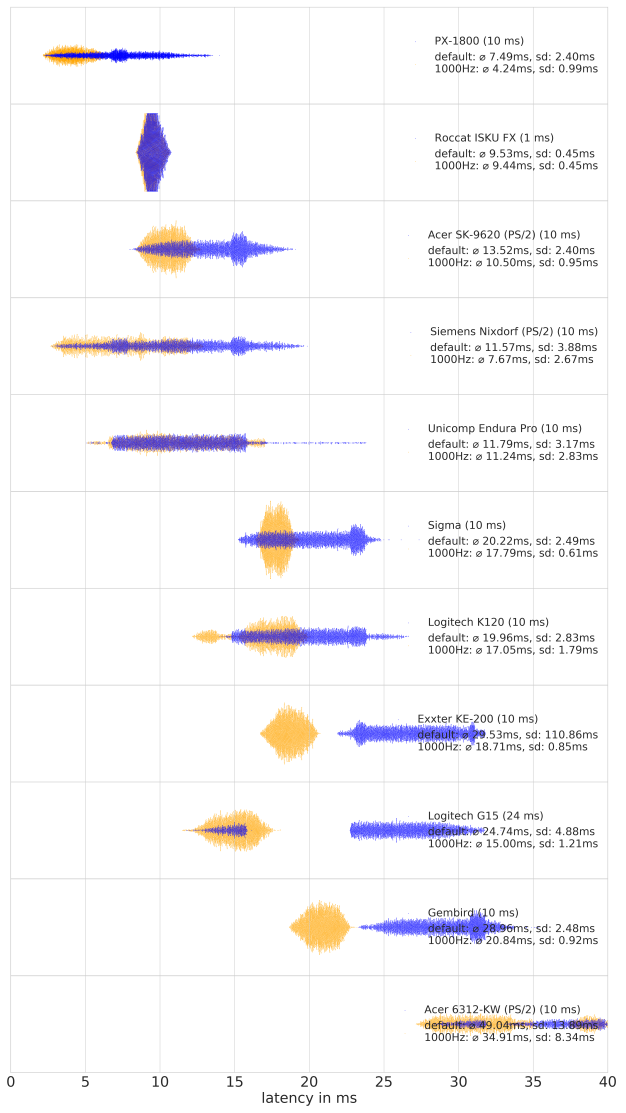

Figure 7: Latency distributions for USB keyboards. Default polling interval is given in the legend. Blue: default polling rate, orange: forced 1000 Hz polling rate.

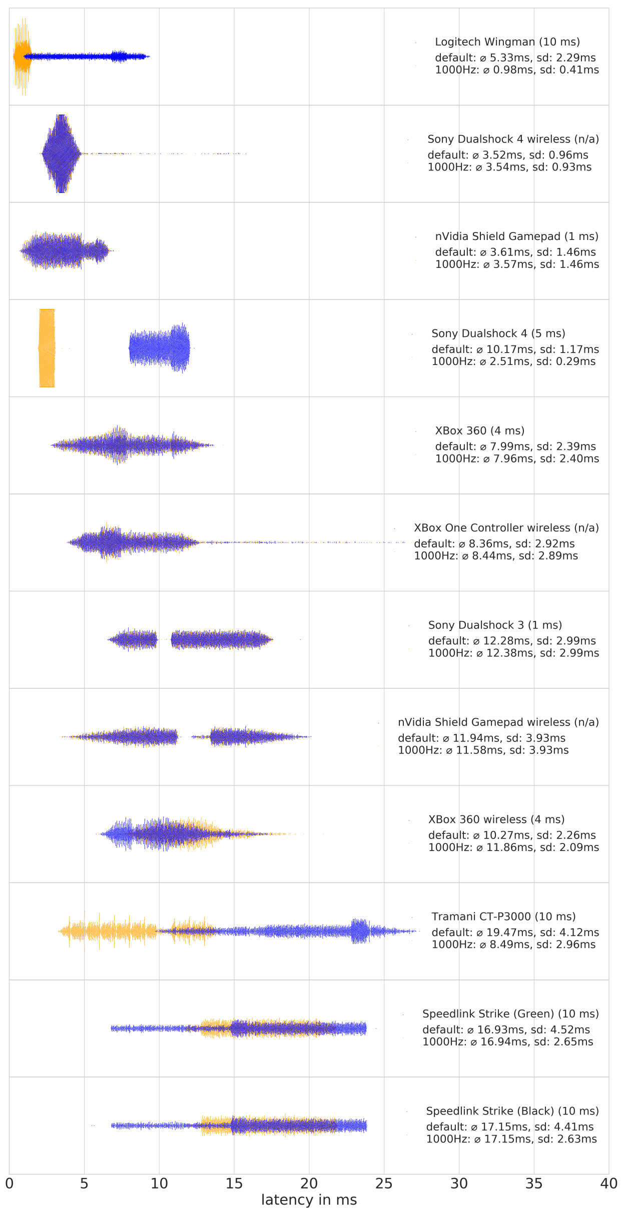

Figure 8: Latency distributions for gamepads. Default polling interval is given in the legend. Blue: default polling rate, orange: forced 1000 Hz polling rate.

For some devices, the latency does not change or changes only little. This is probably because their internal latency is higher than the reported polling rate. For some devices the latency decreases when polled more often, i.e. the internal latency of the devices is lower than their desired polling interval. Most notably, the *Logitech Wingman Precision* - sold for around 10 USD - offers an extremely low latency of 1-2 milliseconds when polled at 1000 Hz.

In summary, forcing a higher polling rate than advertised by the device may reduce latency in some devices. Polling at a higher rate also reduces the variance in latency. We have anecdotal evidence that some devices may not be able to handle higher polling rates. A cheap *GreenAsia* gamepad became permanently unresponsive when polled at 1000 Hz. However, this seems to be a very rare exception. We have not found any evidence online that users who enforced 1000 Hz polling on Linux via *usbhid* parameters temporarily or permanently damaged their devices. Higher polling rates might lead to higher power consumption on the host and on the input device [13]. To test this hypothesis, we measured the power consumption of a Raspberry Pi 2 running an LXDE desktop environment and an attached Logitech RX250 under 2×2 different conditions: with default (125 Hz) and maximum (1000 Hz) polling rate, and while the mouse was sitting still or being jiggled rapidly. Current was measured with milliampere resolution via the power supply. While the mouse was sitting still, current varied between 740 and 750 mA. Rapidly jiggling the mouse resulted in a current of 770-800 mA. In both cases, changing the polling rate did not discernibly increase current consumption. We therefore assume that any increase in power consumption due to higher polling rates is negligible on laptops and desktop computers. Wireless input devices with a USB dongle should not be affected by a higher USB polling rate, as they only send events to the USB dongle when a button/key has been pressed [20]. In our opinion, polling at the maximum rate (usually 1000 Hz) should be enabled in most use cases in order to reduce average latency and latency variance.

6.3 Effect of USB Hubs and Raspberry Pi versions

We conducted measurements with a gamepad connected either directly to the Raspberry Pi, via a cheap USB 2.0 hub, or via a cheap USB 3.0 hub. Measurements were conducted both with the device's default polling rate and with a polling rate of 1000 Hz. None of the measurements indicated that measureable latency was added by a USB hub. We also measured a device's latency using different Raspberry Pi models (1-3) and found no difference.

7 DISCUSSION

In the following, we discuss limitations, findings, and conclusions. Our approach to measuring latency is limited in several ways. As mentioned, LagBox can only measure the latency of USB devices. Currently, only button events are detected. All measurements were conducted on Raspberry Pi devices running Linux. Whether latency on other devices and other operating systems is the same, needs to be investigated. While we made sure that latency sources in our measuring setup are in the low microsecond range, we have not conducted conclusive measurements of kernel scheduling and USB subsystem latency. Also, the latency values we measured are not necessarily an indicator of practical performance. We assume that mechanical latency contributes more to overall input latency than the electrical latency we measured. End-to-end latency might be affected even more by processing latency and output latency.

Nevertheless, our research resulted in a few salient findings. Most importantly, we found that the latencies of most USB input devices have a bimodal or even multimodal distribution. Nearly none of the prior works contain histograms depicting latency distributions. While Casiez et al. show histograms in Figure 5 of their paper, two of them do not show any indications of multimodal distributions [5]. Therefore, the peaks in the lower histogram appear to be caused by timing issues of the Qt framework. Teather et al. report a "roughly bimodal distribution around 12 ms (70%) and 24 ms (30%)" [26]. However, they attribute this effect to timing limitations of their software framework. Damian [7] argues that polling-rate-related variance of latency for typical input devices (with a standard deviation (SD) in the millisecond range) is much smaller than the variation in reaction times (SD in the dozens of milliseconds) for a number of typical psychological experiments. Therefore, effects of latency jitter might usually be neglected. Damian supports this claim with the results of simulations. However, his model assumes that variance is small and follows a uniform distribution. As we show in this paper, this is not the case for most input devices.

8 CONCLUSION AND OUTLOOK

In this paper we have presented a model of input latency, as well as a latency measurements for 36 USB devices. Our measurements show - for example - that great differences exist between input devices, that even for the same device latency may vary up to several dozen milliseconds due to wide latency distributions, and that polling input devices at the maximum speed of 1000 Hz oftentimes improves both average latency and latency variance. Even if input latency is only a small part of end-to-end latency in a computing system, measuring and improving it helps in improving the latency of other components. General concepts discussed in this paper, such as the effect of polling rate or ways of reporting latency, may also be of use in other contexts.

In summary, we suggest that

- partial latencies should be measured as directly as possible in order to better characterize sources of latency,
- that swarmplots should be preferred over other common visualizations and especially over average/SD reports for reporting latency,
- that the issue of input latency in reaction time studies should be revisited,
- and that operating systems should use a polling rate of 1000 Hz for input devices per default at least on desktop and laptop computers.

Future research would include more precisely determining the actual effect of input latency in realistic tasks, extending LagBox to non-binary input events, such as mouse movement, and extending the catalogue of devices whose latency has been measured. LagBox might also be used to calibrate other, non-invasive methods [6]. We informally discussed ethical implications of this research according to the ACM Code of Ethics [1] and do not see any direct or indirect negative effects of our research. Source code and dataset are attached as supplementary material in the ACM Digital Library. Additional information can be found at the accompanying website: hci.ur.de/projects/latency.

ACKNOWLEDGMENTS

This paper builds on earlier work done by Simon Fürnstein, Oliver Pieper, and Mark Engerißer in their bachelor's theses under the supervision of Raphael Wimmer. We would like to thank all reviewers for their helpful feedback, Sunjun Kim for valuable discussions, the members of the linux-input mailing list for background information on the USB polling rate, and Lucia Eckl for editing the accompanying video. This project is funded by the Bavarian State Ministry of Science and the Arts in the framework of the Centre Digitisation.Bavaria (ZD.B)

REFERENCES

[1] ACM Code 2018 Task Force. 2018. ACM Code of Ethics. Retrieved September 19, 2018 from https://www.acm.org/code-of-ethics

[2] Christiane Attig, Nadine Rauh, Thomas Franke, and Josef F. Krems. 2017. System Latency Guidelines Then and Now – Is Zero Latency Really Considered Necessary?In *Engineering Psychology and Cognitive Ergonomics: Cognition and Design* (Lecture Notes in Computer Science), 3–14.

[3] Florian Bockes, Raphael Wimmer, and Andreas Schmid. 2018. LagBox – Measuring the Latency of USB-Connected Input Devices.In *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems* (CHI EA '18), LBW115:1–LBW115:6. https://doi.org/10.1145/3170427.3188632

[4] Stuart K Card, Thomas P Moran, and Allen Newell. 1986. The model human processor - an engineering model of human performance.*Handbook of perception and human performance.* 2, 45–1.

[5] Géry Casiez, Stéphane Conversy, Matthieu Falce, Stéphane Huot, and Nicolas Roussel. 2015. Looking through the Eye of the Mouse: A Simple Method for Measuring End-to-end Latency using an Optical Mouse.629–636. https://doi.org/10.1145/2807442.2807454

[6] Géry Casiez, Thomas Pietrzak, Damien Marchal, Sébastien Poulmane, Matthieu Falce, and Nicolas Roussel. 2017. Characterizing Latency in Touch and Button-Equipped Interactive Systems.In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology* (UIST '17), 29–39. https://doi.org/10.1145/3126594.3126606

[7] Markus F. Damian. 2010. Does variability in human performance outweigh imprecision in response devices such as computer keyboards?*Behavior Research Methods* 42, 1: 205–211. https://doi.org/10.3758/BRM.42.1.205

[8] Jonathan Deber, Bruno Araujo, Ricardo Jota, Clifton Forlines, Darren Leigh, Steven Sanders, and Daniel Wigdor. 2016. Hammer Time!: A Low-Cost, High Precision, High Accuracy Tool to Measure the Latency of Touchscreen Devices.2857–2868. https://doi.org/10.1145/2858036.2858394

[9] Ricardo Jota, Albert Ng, Paul Dietz, and Daniel Wigdor. 2013. How fast is fast enough?: A study of the effects of latency in direct-touch pointing tasks.In *Proceedings of the sigchi conference on human factors in computing systems*, 2291–2300.

[10] Topi Kaaresoja and Stephen Brewster. 2010. Feedback is... Late: Measuring multimodal delays in mobile device touchscreen interaction.1. https://doi.org/10.1145/1891903.1891907

[11] Sunjun Kim, Byungjoo Lee, and Antti Oulasvirta. 2018. Impact Activation Improves Rapid Button Pressing.In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (CHI '18), 571:1–571:8. https://doi.org/10.1145/3173574.3174145

[12] Ulrich Lampe, Qiong Wu, Ronny Hans, André Miede, and Ralf Steinmetz. 2013. To Frag or to Be Fragged - An Empirical Assessment of Latency in Cloud Gaming:5–12. https://doi.org/10.5220/0004345900050012

[13] Linux Input Mailing List. 2018. Re: Reasons for respecting bInterval? Retrieved January 7, 2019 from https://www.spinics.net/lists/linux-input/msg59274.html

[14] Linux Kernel Developers. 2019. Linux kernel: Drivers/usb/core/urb.c. Retrieved January 7, 2019 from https://github.com/torvalds/linux/blob/master/drivers/usb/core/urb.c#L566

[15] Dan Luu. 2017. Terminal latency. Retrieved January 7, 2019 from https://danluu.com/term-latency/

[16] Dan Luu. 2017. Keyboard latency. Retrieved January 7, 2019 from https://danluu.com/keyboard-latency/

[17] Dan Luu. 2017. Computer latency: 1977-2017. Retrieved January 7, 2019 from https://danluu.com/input-lag/

[18] I Scott MacKenzie and Colin Ware. 1993. Lag as a determinant of human performance in interactive systems.In *Proceedings of the interact'93 and chi'93 conference on human factors in computing systems*, 488–493.

[19] Microsoft. _USB_ENDPOINT_DESCRIPTOR documentation. Retrieved January 7, 2019 from https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/content/usbspec/ns-usbspec-_usb_endpoint_descriptor

[20] Marc Newlin. 2016. MouseJack - Injecting Keystrokes into Wireless Mice. Retrieved January 10, 2019 from https://github.com/BastilleResearch/mousejack/blob/master/doc/pdf/MouseJack-whitepaper-v1.1.pdf

[21] Albert Ng, Julian Lepinski, Daniel Wigdor, Steven Sanders, and Paul Dietz. 2012. Designing for low-latency direct-touch input.In *Proceedings of the 25th annual acm symposium on user interface software and technology* (UIST '12), 453–464. https://doi.org/10.1145/2380116.2380174

[22] Antti Oulasvirta, Sunjun Kim, and Byungjoo Lee. 2018. Neuromechanics of a button press.4099–4112. https://doi.org/10.1145/3173574.3174082

[23] Richard R. Plant. 2016. A reminder on millisecond timing accuracy and potential replication failure in computer-based psychology experiments: An open letter.*Behavior Research Methods* 48, 1: 408–411. https://doi.org/10.3758/s13428-015-0577-0

[24] Richard R. Plant, Nick Hammond, and Tom Whitehouse. 2003. How choice of mouse may affect response timing in psychological studies.*Behavior Research Methods, Instruments, & Computers* 35, 2: 276–284. https://doi.org/10.3758/BF03202553

[25] Reiza Rayman, Serguei Primak, Rajni Patel, Merhdad Moallem, Roya Morady, Mahdi Tavakoli, Vanja Subotic, Natalie Galbraith, Aimee Van Wynsberghe, and Kris Croome. 2005. Effects of latency on telesurgery: An experimental study.In *International conference on medical image computing and computer-assisted intervention*, 57–64.

[26] Robert J. Teather, Andriy Pavlovych, Wolfgang Stuerzlinger, and I. Scott MacKenzie. 2009. Effects of tracking technology, latency, and spatial jitter on object movement.43–50. https://doi.org/10.1109/3DUI.2009.4811204

[27] USB Implementers Forum. 2000. Universal Serial Bus Specification Revision 2.0. Retrieved from https://www.usb.org/document-library/usb-20-specification

[28] USB Implementers Forum. 2001. Universal Serial Bus (USB) Device Class Definition for Human Interface Devices (HID), Firmware Specification Version 1.11. Retrieved January 7, 2019 from https://www.usb.org/sites/default/files/documents/hid1_11.pdf