MousePath: Enhancing PC Web Pages through Smartphone and Optical Mouse

Zhiwei Wang†, Qianyi Huang‡, Yihui Yan†, Haitian Ren†, Yizhou Zhang†, Zhice Yang†
†*School of Information Science and Technology, ShanghaiTech University*
‡*Southern University of Science and Technology, and Peng Cheng Laboratory*
†{wangzhw, yanyh, renht, zhangyzh, yangzhc}@shanghaitech.edu.cn, ‡huangqy@sustech.edu.cn

***Abstract*—This paper proposes MousePath, a novel lightweight communication system between PC web pages and smartphones. MousePath works by putting the optical mouse on top of the smartphone's screen, then its transmission starts and is instantly finished without association and pairing friction. It encodes data into the movement of smartphone's display content and leverages the optical mouse of the computer to sense the movement for decoding the data. We prototype and evaluate the system with commercial computers and smartphones. A key benefit of MousePath is that it can be seamlessly integrated into web pages. Two representative web applications, *i.e.*, sensor sharing and message sharing, have been developed to demonstrate MousePath's potential in enhancing PC web page functionalities.**

***Index Terms*—Cross Device Information Sharing; Optical Mouse; Visible Light Communication**

I. Introduction

The Web is progressively evolving. Traditional web pages primarily serve as an output interface for users to retrieve information from a remote server, while recent web pages also try to retrieve client-side information to enrich its functionalities. For instance, Media Capture and Streams API [1] enables web VoIP through accessing microphones and cameras.

Motivated by the popularity of smartphones, this paper considers a question following the above trend: can PC web pages, *i.e.*, web pages accessed via a desktop or a laptop, be further enhanced by taking information from co-located smartphones? A positive answer would lead to several interesting and beneficial web applications. For example, it will allow the PC web page to retrieve the account and password credentials stored in the smartphone for the auto-filling in of the login session. Another example is sensor sharing. Smartphones are rich in various sensing capabilities. Unique ones can be made use of by PC web pages. Below are a few examples. Accessing the localization sensors of the smartphone enables precise location-based services (LBS) on PC web pages. Through light sensors, the online photo editor can auto-tune its color space to fit the ambient light condition. Further, accessing the heartrate sensor is helpful for remote health diagnosis.

When putting the above idea into practice, a data connection between the smartphone and the PC web page must be established. However, this is surprisingly challenging. There are already mature solutions for sharing information between smartphones and PCs, such as the Your Phone App in Window 10 and the General Clipboard in Apple products. They all target *phone-to-PC* sharing rather than *phone-to-web* sharing.

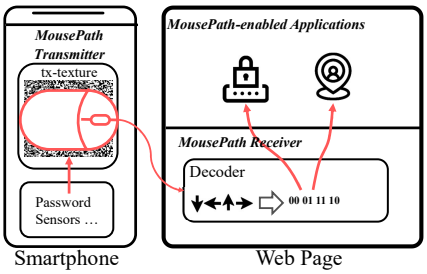


Fig. 1. Overview of MousePath. MousePath consists of two major entities. The *MousePath transmitter* app in the smartphone and the *MousePath receiver* script embedded in the web page. The *MousePath transmitter* gathers information in the smartphone, like account credentials and sensor data, and transfer them to the *MousePath receiver* to enable new web functionalities, *e.g.*, enabling Precise Location-based Service, and facilitating Password Managers.

In situations where the web page already provides suitable input interfaces, such as text box and image drop area, the two concepts are close. However, when the web page requires verbose input such as location attitudes and raw sensor samples, there is a gap between the two concepts. The *phone-to-PC* sharing brings only information to the PC's operating system, thus additional effort, *i.e.*, installing a browser plugin, is needed to further convey the information to the web page. This, however cancels out the major benefits of using a web application - the simplicity of no installation or management efforts being required.

In this paper, we propose MousePath, a lightweight and ubiquitous way to realize *phone-to-web* sharing. As shown in Figure 1, it works by putting an optical mouse on top of a smartphone screen, and then the information from the smartphone is directly transferred to the web application.

MousePath transfers data through a novel channel lying between the smartphone screen and the PC's optical mouse. The key insight is to make use of the motion sensing ability of optical mice. We observe that, when putting it on the screen of the smartphone, the optical mouse has the capability of sensing the movements of the display content. Based on this property, the MousePath transmitter in the smartphone encodes the data into the movements of the display content, which fools the optical mouse into treating the content movement as real physical movement. As a result, the MousePath receiver, *e.g.*, a piece of JavaScript in the web page, can infer the movements of the display content from the system's mouse trajectories, through which the data can be decoded. The above scheme represents a special screen-to-mouse communication channel.

Although we have not noticed any other *phone-to-web* sharing systems, MousePath is not the only way to achieve this. For example, by using the latest Media Capture and Streams API [1], and compatible browsers, web pages can access the camera to capture a QR code stream on the smartphone to achieve similar functionalities. In MousePath, the optical mouse is like a camera and the moving display content of the smartphone is like the QR code. MousePath is superior over the camera-based method in API compatibility, hardware availability, and visual privacy.

Compared with potential ad-hoc radio-based approaches, such as Bluetooth, MousePath has the following advantages. First, MousePath is pairing-free. Similar to near-field-communications (NFC), the communication entities identify each other through physical proximity, *i.e.*, just put the mouse on top of the phone, with no other user burden. This feature is particularly useful when using it with public PCs, where pairing raises privacy concerns and might not even be allowed. Second, MousePath implies benefits in security. The physical proximity largely eliminates wireless sniffing [2], hence the user can place more trust in the channel, *e.g.*, to transfer the credentials. Further, as a single-way channel, MousePath provides strong isolation to protect phones against malicious PCs (*e.g.*, in public places) [3].

Our contributions are:

- We propose a novel lightweight screen-to-mouse communication channel by leveraging screens for transmitting and optical mice for receiving. To our knowledge, there are no similar methods which utilize the original sensing ability of optical mice for communication.
- We implement a prototype system of MousePath and evaluate it with various commodity optical mice and smartphones. The receiver is written in Javascript and can be embedded into web pages and run with unmodified web browsers.
- We demonstrate MousePath with two web applications. MousePath enables precise location-based services for PC web pages and also shows the advantage to integrate with password managers.

II. Related Work

Screen-to-camera Channel. Since the optical mouse sensor is actually an image sensor, MousePath is related to the area of screen-to-camera communication. At a high level, the approaches encode information in the display content of the screen, and use a camera to capture the screen and decode the information. The basic example is scanning QR-codes. Recent efforts in screen-to-camera research improve the performance in various aspects, including the data rate [4], computation overhead [5], *etc.* Unlike the aforementioned, the screen-to-mouse channel leveraged in this paper has not been investigated before. Specifically, since the optical mouse by default only provides movement information, MousePath decodes messages from the mouse's moving trajectories instead of from images. Further, MousePath encodes messages by slightly shifting the same display pattern in consecutive display frames instead of modifying the display content.

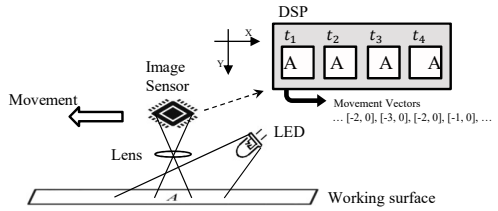


Fig. 2. Optical Mouse Uses Image Sensor to Detect Movement. For example, when the mouse is moved leftwards, the mouse DSP detects the movement according to the rightward movement of the texture "A" in the captured images. The mouse reports its observation via movement vectors showing negative X values, *i.e.*, the leftward direction in the mouse's local coordinate system.

Enhancing Desktop with Smartphones. There is a long history of researchers combining smartphones and desktops. The Pebbles project [6] makes use of a touch-screen PDA to display the user interface of desktop applications, providing an additional display and control interface. Similar approaches are discussed under the context of data sharing [7], math equation editing [8], and so on. Moreover, smartphones are explored as a trusted computing device to improve the security of desktop computers [9]. All of these works assume there is a network connection between the smartphone and the desktop, but ignore the practical overheads in establishing and maintaining the connection. MousePath provides a ubiquitous and convenient way to transfer data from a smartphone to a desktop.

III. Background

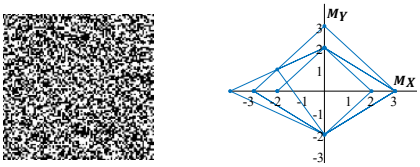
This section provides the background knowledge of the techniques used in MousePath.

A. Optical Mouse

A mouse is a major way for computer users to interact with the Graphical User Interface (GUI) system. A mouse works by continuously tracking its relative movement on the working surface. The GUI system scales movement values and updates the cursor location which allows the user to control the cursor to point, select, and drag virtual objects on the screen.

An optical mouse is a kind of optical imaging system (Figure 2). It uses a backlight (LED or laser) to illuminate the working surface. The image sensor, which is actually a low resolution but high frame rate CMOS sensor, continuously takes images of the working surface at a rate of thousands of frames per second. When the mouse is moved, the images change as the textures of the captured working surface also change (see the anchor character "A" in Figure 2 for example). The sampling rate of the image sensor is so high that sequential images tend to partially overlap. Thus, the Digital Signal Processing (DSP) unit of the optical mouse can make use of the overlapped images to compute the direction and distance of the movement [10].

The DSP quantifies the movements to *movement vectors* $\vec{M}(t_i) = [M_X(t_i), M_Y(t_i)]$, which represent the accumulated displacement along the X-axis and Y-axis of a mouse's local coordinate system from time t_{i-1} to time t_i. Note integration



(a) (b)

Fig. 3. Optical Mouse Can Detect the Movement of the Display Content of the Underlying Screen. The optical mouse and smartphone are put together as shown in Figure 1. Subfigure (a) is the texture shown on phone's screen. When the texture is shifted in clockwise order and the mouse is not physically moved, movement vectors reported by the optical mouse are shown in (b). The dots and lines indicate that the movement of the display content can be correctly detected by the optical mouse.

of on $\vec{M}(t_i)$ gives the trajectory of the movements, thus $\vec{M}(t_i)$ are reported to the host system for GUI control. Although, in principle, the DSP can generate one $\vec{M}(t_i)$ for every two sequential images, the default reporting/sampling rate of $\vec{M}(t_i)$ is fixed at 125 Hz in normal optical mice.

1) Idea of Screen-to-Mouse Channel: MousePath enables a generic and ubiquitous communication channel between public computers and smartphones. Its approach is based on the following observation; *besides the reflected backlight, the image sensor of the optical mouse can also capture other light signals*. Specifically, when the optical mouse is put on a screen, it is possible to capture the display content.

Motivated by the above observation, an interesting question is, will the optical mouse report movements if the underlying display content moves virtually but the mouse itself is not moving? Figure 3 answers this question through a simple experiment. An optical mouse is put on the screen of a smartphone as seen in Figure 1. The overlapped area of the screen displays a texture as shown in Figure 3(a), which covers the sensing area of the optical mouse Each time when the phone's display updates (typically 60 Hz), the texture is shifted by 2 pixels to one of the four directions in clockwise order, *i.e.*, up-right-down-left-up. The reported motion vectors $\vec{M}(t_i)$ are plotted in Figure 3(b). Sequential vectors are connected by lines. The dots and lines in Figure 3(b) form a circle coinciding with the clockwise shift/movement of the texture, indicating a positive answer to the question.

The above observation and validation imply an untapped opportunity in transferring data from the smartphone to the computer by fooling the optical mice with the movement of the smartphone's display content. Specifically, the data can be encoded in the shift directions of the texture. At the same time, the optical mouse can identify the shift directions, and report to its host system with movement vectors. Applications in the host system can then decode the data encoded by the smartphone through analyzing the movement vectors.

B. MousePath Overview

The overview of the MousePath system is shown in Figure 1. It consists of two major entities, the *MousePath transmitter* app and the *MousePath receiver* web script.

The *MousePath transmitter* is an app running in the user's smartphone (Section IV-A). The app displays a *tx-texture* on its GUI and shifts it to stimulate the optical mouse to generate movement vectors. The shift directions and distance are chosen according to the modulation method and the data bits. The app is also responsible for managing the data to be transmitted, for example, the login credentials, data from sensors, and the like.

The *MousePath receiver* is a web script offered by the webserver and running in the web browser on the PC. The core part of *MousePath receiver* is the decoder (Section IV-B). The decoder routine obtains movement vectors from the system's mouse interface and decodes them to obtain the bit stream transmitted by the *MousePath transmitter*.

IV. Screen-to-Mouse Channel

This section extends the idea in Section III-A to realize a reliable screen-to-mouse channel. This special communication channel, which has not been studied before, contains unique challenges rooted from both the smartphone's display system and the optical mouse's imaging system. We address them with the transmitter and receiver design.

A. Transmitter Design

This subsection describes the modulation, synchronization, channel coding designs used in the *MousePath transmitter*.

1) Modulation: The movement of the texture in the *MousePath transmitter* app can stimulate the reaction of the optical mouse sensor. The displayed texture, called *tx-texture*, and its movements determine the intensity of the reaction, *i.e.*, the amplitude of the movement vector $\vec{M}(t_i)$, and thus affect the quality of the channel. Therefore, the design goal of *tx-texture* and its movement pattern is to maximize the amplitude of the mouse movement vectors while avoiding possible movement ambiguities. There are two properties complicating the problem.

First, optical mice have a maximum detectable moving speed, which is related to the sampling rate of the image sensor. Recall that the mouse DSP measures movements by comparing consecutive images, hence the displacement between two images should not be too large, otherwise the consecutive images will differ too much to correctly judge the displacement. The maximum moving speed of evaluated mice is limited to 30 inches per second [11]. As the image sensor samples 3000 images per second, the maximum displacement of the underlying texture between two sequential mouse images should not exceed $30/3000 = 0.01$ in, which is equal to the width of 4 pixels of the screen (400 pixels per inch).

The movement emulated by the *tx-texture* is quite different from the real physical movement. As the screen refresh rate (60 Hz) is much slower than the mouse's sampling rate (3000 Hz), from the perspective of the optical mouse sensor, its images are identical most of the time. As a result, the maximum shifts of the *tx-texture* is determined by the two images encountering the screen refresh rather than all images during the $1/60$ s. Therefore, the shift distance of the *tx-texture* in two sequential display frames should be within 4 pixels.

Second, the mouse's imaging system can only cover an area of 0.04 in $\times$ 0.04 in of the working surface [11]. When the mouse is placed on the phone's screen (the pixel density

is about 400 pixels per inch). The optical mouse judges the movements according to an area of 16 pixels $\times$16 pixels of the *tx-texture*.

The above properties indicate that, in order to uniquely determine the shift direction, for any 16 pixels $\times$16 pixels area in the *tx-texture*, there should be no neighboring areas within 4 pixels that are identical or similar. We generate *tx-texture* based on this constraint. In addition, we use black and white textures to increase the contrast for the image sensor (see examples in Figure 3(a)).

Similar to the modulation methods in other communication systems [12], both the shift direction and shift distance of the *tx-texture* can be used to represent bits. However, as the channel contains large noise, we fix the shift distance and only modulate the shift directions. We call the scheme *Direction Shift Keying* (DSK). Table I shows one possible mapping, which uses four shift directions as symbols ($\rightarrow, \downarrow, \leftarrow, \uparrow$) to represent bits. The symbol "•" represents no shift for the display frame, which is mainly used in the preamble.

TABLE I
MODULATION: DIRECTION SHIFT KEYING

Bits	00	01	10	11	null
Symbol	$\downarrow$	$\leftarrow$	$\rightarrow$	$\uparrow$	•

2) Preamble: Each packet has a preamble, which is used to identify the beginning of a packet. The preamble can also be used to find the rough boundary of symbols. We use synchronization in Section IV-B to determine the fine boundary. Specifically, the preamble uses the following symbol sequences:

$$\leftarrow\leftarrow \bullet\bullet \rightarrow\rightarrow \bullet\bullet$$

3) Increasing Reliability with Channel Coding: Despite the adoption of conservative modulation schemes, in practice, the bit error rate is still much higher than the theoretical expectation. The reason is partially analyzed in the previous modulation scheme. The optical mouse judges texture movement according to images taken when display frames switch. However, if the image sensor samples at the transition stage of two display frames, they may capture unstable frames [13]. In such cases, the image sensor might report inaccurate or even the wrong direction, corrupting that symbol. To increase the successful delivery rate of messages under such harsh conditions, we use Reed-Solomon (RS) codes [14] for forward error correction. Each packet consists of a preamble and payload field. The payload is first coded to RS-blocks and then mapped to direction shift symbols for transmission. CRC-16 is used to detect errors in each RS-block.

B. Receiver Design

This subsection first characterizes the receiving properties of the optical mouse, then describes how the *MousePath receiver* decodes information from movement vectors.

1) Receiving Properties of the Optical Mouse: Figure 4 demonstrates an example of received movement vectors. According to Section III, the mouse should only report movement when the display frame refreshes. However, as shown in Figure 4, in Slot 2, 4, 6, even though the screen is static during the sampling period, the mouse still reports non-static movement vectors. This is probably due to the smoothing function of the mouse's DSP, *i.e.*, the mouse reports movement consistent with the previous sampling period. Due to this reason, the first samples reported by the mouse after the screen refreshes are usually more accurate.

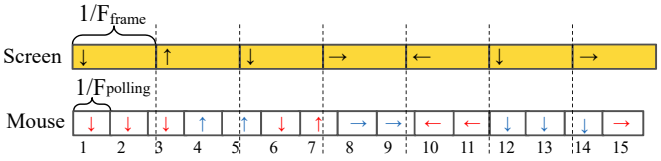


Fig. 4. Sampling Offset and Noise Exist in Received Movement Vectors. The screen refresh rate ($F_{\text{frame}} = 60$ Hz) and the mouse reporting rate ($F_{\text{polling}} = 125$ Hz) are different, bringing in a large sampling offset (see movement vectors reported at Slots 3 and 14). The movement vectors also contain noise or even wrong values (see Slot 7).

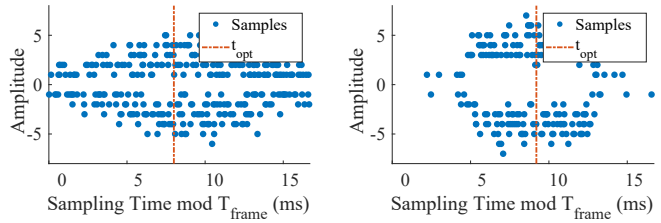


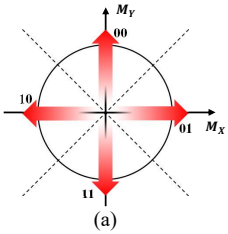
Fig. 5. Determine the Optimal Sampling Time. Dots in the figure are amplitudes of the X movement vectors folded in one refresh cycle T_{frame}, *i.e.*, [x,y] = [t_i mod T_{frame}, $M_X(t_i)$]. As the mouse reports every 8 ms, samples in the range $[t_{\text{opt}} - 4\text{ms}, t_{\text{opt}} + 4\text{ms}]$ mod T_{frame} are the most desirable, which are less noisy than the others. (a) Samples from mouse DELL MO56UOA. (b) Samples from mouse DELL MS111.

2) Preamble Detection: The receiver continuously correlates the receiving movement vectors with the preamble sequence. A preamble is detected if the correlation energy is higher than a threshold. Then, the receiver obtains the rough start of the data symbols and is ready for fine-tuning the re-sampling time.

3) Re-sampling at the Optimal Sampling Time: Since for each screen refresh, the first reporting event is more accurate, we want to identify them for decoding. However, as the screen and the mouse are not synchronized, the receiver does not know the exact time the screen refreshes.

We infer the screen refresh time by observing the statistics of the reported amplitude. When the reporting event is the first one after the screen refresh, its amplitude is usually large, *e.g.*, above 4 or below -4, where positives and negatives indicate the direction; when the reporting event is not the first one, its amplitude is usually smaller, *e.g.*, between -2 and 2.

We take advantage of the eye diagram [15] to show this effect. Figure 5 is the eye diagram of the X movement vectors, which collectively shows vectors reported by the mouse of a certain period by folding their sampling timestamps into $[0, T_{\text{frame}}]$. We can search within this range to find the value t_{opt} where samples in $[t_{opt} - 4\text{ms}, t_{opt} + 4\text{ms}]$ have a relatively larger amplitude. This is because t_{opt} reflects the time at which the screen refreshes. We choose samples close to t_{opt} mod T_{frame} for demodulation.



(a)

Movement Vector $\vec{M} = [M_X, M_Y]$	Bits
$M_Y > 0$ && $\lvert M_Y \rvert > \lvert M_X \rvert$	00
$M_X > 0$ && $\lvert M_X \rvert \geq \lvert M_Y \rvert$	01
$M_X < 0$ && $\lvert M_X \rvert \geq \lvert M_Y \rvert$	10
$M_Y < 0$ && $\lvert M_Y \rvert > \lvert M_X \rvert$	00

(b)

Fig. 6. Demodulation of Direction-shift Keying. The receiver maps the movement directions into bits. The direction is determined by the polarity and amplitude of the movement vector $\vec{M} = [M_X, M_Y]$. The visual illustration is shown in (a), which divides the 2D space into four regions. The lookup table is shown in (b).

4) Demodulation: The demodulation is an inverse process of modulation. The movement vectors are mapped to bits according to Figure 6(b). As the directions of the reported movement vectors are opposite to the shift direction of *tx-texture*, Table I and Figure 6(b) are opposite. Note that the orientation of the phone and the mouse might not be accurately aligned, and we evaluate how the misalignment angle affects the decoding performance in Section VI.

V. Implementation

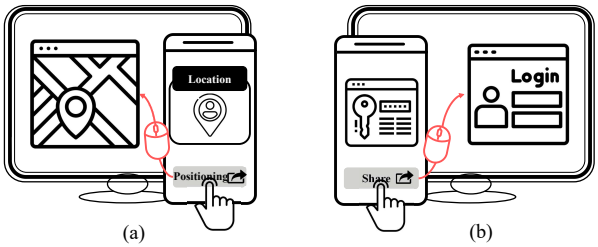
We first give two examples to show how MousePath can be used in practice, then describe the implementation details.

A. MousePath Enabled Applications

To demonstrate the potential of applying MousePath to enhance PC web page functionalities, we implement two web applications shown in Figure 7 and described below.

1) Precise Location-based Service (LBS): When using web maps, online shopping, *etc.*, precise location allows for a better user experience, *e.g.*, auto-filling the shipping address and recommendation of nearby coupon events. However, common indoor PCs lack dedicated localization sensors. They mainly rely on IP addresses to determine coarse-grained locations at the building or street-block level (GeoIP). In regions relying on NATs for Internet access, location errors can span to hundreds of meters. On the contrary, the localization approaches of today's smartphones are more diverse and accurate. When GPS is not available indoors, many of its attributes like the locations of cellular base stations and Wi-Fi access points can be taken advantage of [16]. Smartphones supporting Wi-Fi RTT [17] can even be localized at the room level. MousePath enables precise LBS for PC web pages by retrieving the location information from co-located smartphones.

2) Password Manager Companion: Today's web passwords are complicated and are a burden for people to remember. One popular mitigating way is to store credentials in one place, such as password managers [18], which lock passwords in a USB dongle or cloud and do auto-filling via a simple click or copy-pasting action. However, it is not always convenient to use password managers across devices. For example, in public PCs, such as consoles in libraries, the USB interface is usually blocked and the login process to the cloud is subject to the risk of leaking the master key of the manager. MousePath allows the web page to retrieve credentials stored in the smartphone without the above limitations[1].



(a) (b)

Fig. 7. MousePath-enabled Web Applications. (a) Precise LBS: PCs lack precise localization sensors, whereas smartphones do not. MousePath enables precise LBS on PC web pages through retrieving the location information from a co-located smartphone. (b) Password Manager Companion. The login credentials stored in the smartphone are shared to the web page for auto-filling.

B. MousePath Transmitter

The MousePath transmitter app calls the C++ Reed-Solomon library through Java Native Interface (JNI) for data encoding. The Galois field $GF(2^6)$ is used in Reed-Solomon coding. In order to modulate the movement vectors, the MousePath transmitter app displays a 500×500 pixels *tx-texture* on the screen. The *tx-texture* is updated 60 times per second. An example of the MousePath transmitter app UI is shown in Figure 8(b). The location data or the password are encoded in the *tx-texture*. Then the user can put a mouse on top of the *tx-texture* to start the data transmission.

In Android, the display pipeline is synchronized by the VSync signal. When the VSync comes, the UI will display the previous frame if the rendering of the current frame has not been finished. This phenomenon is called Jank [19]. To avoid Jank, the transmitter app uses the GPU instead of the CPU for rendering, and binds the *tx-texture* to the GPU texture cache. This ensures that the rendering latency is within 16 ms.

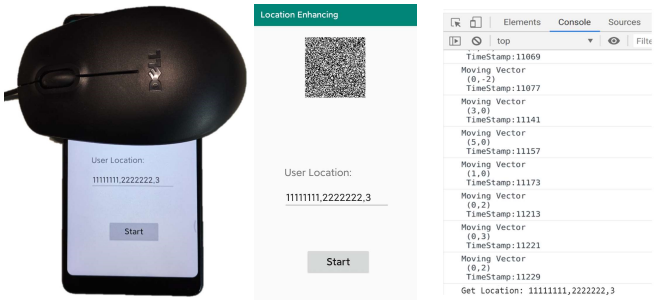
C. MousePath Receiver

The MousePath receiver hardware includes an optical mouse and a PC. We implement the MousePath receiver as a Javascript application so that it can be embedded into web pages. The MousePath receiver uses the `Mouse Lock API` [20] to constrain the target mouse cursor within an HTML widget to prevent the cursor from going past the boundary of the window. The MousePath receiver captures and demodulates the movement vectors from the `mousemove` events. Then the MousePath receiver uses the JavaScript Reed-Solomon library for error correction. As shown in Figure 8(c), the script can successfully decode the information.

VI. Evaluation

In this section, we present the evaluation results. We evaluate the performance of MousePath from the communication aspect and its effectiveness in the two web applications.

[1]Note that the path to the PC can be further secured through encrypting the MousePath channel with a one-time key, *e.g.*, sent from the web server to the smartphone through Short Message Service (SMS).



(a) (b) (c)

Fig. 8. Implementation of MousePath. (a) Putting the mouse on the smartphone to use it. (b) UI for location sharing. (c) The example output from the browser's console, showing movement vectors and timestamps.

A. Evaluation of Communication Performance

1) Texture-related factors: We first evaluate how the texture-related parameters affect the MousePath performance. We mainly focus on the following three parameters: First, the movement distance of the *tx-texture* in consecutive frames, which is called **shift step** s. Second, the size of the minimum color blocks in the *tx-texture*, which is called **texture granularity** g. Third, the **angle** between the mouse and the *tx-texture* when the mouse is placed on the *tx-texture*. By default, we use a Xiaomi Mix 2S smartphone as the transmitter and a Logitech M100 optical mouse as the receiver. We test these parameters using the symbol error rate (SER) as the metric. Each SER value is calculated over $10,000$ symbols.

First, we evaluate the MousePath with different combinations of shift step s and texture granularity g. The texture we use is a randomly generated one. Figure 9(a) shows the average SER for s from 1 pixel to 4 pixels and g from 2×2 pixels to 4×4 pixels. In all these cases, the SERs first decrease and then increase. As mentioned in Section IV-A, the maximum value of s should be less than 4 pixels. From Figure 9(a), we can see that 2 and 3 pixels are the most suitable values for s as 1 pixel is too small to trigger the movement. Similar to s, a large or small g is not good for the performance. Therefore, we set s to be 2 pixels and g to be 3×3 pixels as the default parameters in MousePath.

Next, we evaluate the impact of angle alignment. We define the correct usage of the MousePath as being to put the mouse on top of the screen and parallel to the short edge of the screen, as shown in Figure 8(a). However, in practice, there may be an angle between the *tx-texture* and the image sensor. Figure 9(b) plots the CDF of SER versus the angle. When the angle increases, the performance becomes worse. When the angle is 20° or above, moving upward can easily be interpreted as moving leftward or rightward, and thus the SER is high; when the angle is 10° or less, with a high probability the SER is less than 10%. According to our experience, when the user places the mouse, the angle is usually within 10°.

2) Device-related Factors: In this part, we evaluate the performance of MousePath on different smartphones and mouse models. We choose Xiaomi Mix 2S (with an LCD screen), Xiaomi 8 (with an OLED screen) and Huawei Mate 20 (with an LCD screen). For the optical mouse, we choose DELL MS111, DELL MO56UOA and Logitech M100 as the receiver. Figure 9(c) plots the CDF of SER for different smartphone-mouse combinations. The performance of the optical mouse is different, as different models have different sensitivities in their detecting algorithm. From Figure 9(c), we can see that the model with the best performance on Xiaomi Mix 2S is DELL MS111. Figure 9(c) also shows the impact of different screens. We notice that the performance difference between LCD and OLED is not obvious. Besides, the Xiaomi smartphone has better performance than Huawei, because the screen refresh rate is not stable in the Huawei model.

TABLE II
LOCALIZATION ERROR W/ AND W/O MOUSEPATH

Localization Method	Localization Error (m)			
	Dormitory	*Canteen*	*Library*	*Laboratory*
IP Based	291	365	433	380
Smartphone's Built-in	42	19	5	11

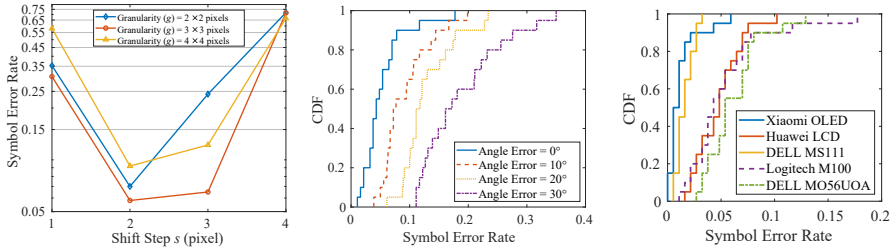
B. Evaluation of MousePath Enabled Applications

This subsection evaluates how effective applying MousePath is to enhance the two web applications.

Application I: Precise Location-based Service. In the transmitter app, we extract the location estimation of the smartphone through the Android location API and share it to the PC web page through MousePath. By default, the PC web page uses the Geo-IP database [21] for getting the estimated location. We sample locations in different places and compare the accuracy of the two approaches in Table II. In all the locations, IP-based localization is not accurate. This is because the resolution of the Geo-IP database in our area is blurred by NAT. On the contrary, the location obtained from the smartphone is more accurate since it uses information from, *e.g.*, the locations of observable base stations.

Application II: Password Manager Companion. With MousePath, a smartphone can share passwords to the PC web page by coding the information in *tx-texture*. We measure the delay of transmitting a password of 20 bytes and compare MousePath with other network technologies. The overall delay includes the device pairing delay, the user interaction delay, and the data transmission delay. Device pairing delay is the time required to set up a connection between the smartphone and the PC. Interaction delay is the time users spend on interacting with the two devices. The transmission delay is the time required for data transmission.

The detailed setup is: For Bluetooth Low Energy (BLE), we use the built-in Bluetooth interface in the smartphone and the PC. For Local Area Network (LAN), we use the third-party software `KDE Connect`. For Wi-Fi, we choose `SHAREit`. The PC acts as a Wi-Fi access point and the smartphone connects to the PC as a client. For MousePath, sharing the password does not require device pairing. The interaction includes selecting MousePath as the sharing interface in the mobile app and placing the mouse on the screen. We note due to the low bit rate, its transmission delay is greater than other wireless technologies. For each test, the results are averaged over 10 measurements by three participants ($P1$, $P2$, $P3$). All



(a) Shift Step and Texture Granularity (b) Bearing Angle Error (c) Different Screens and Mice

Fig. 9. Error Rate v.s. Different Configurations.

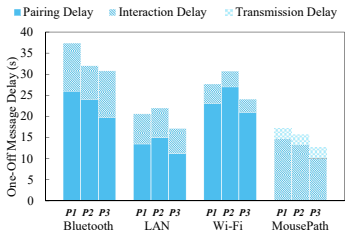


Fig. 10. Delay measured when sharing a password of 20 bytes to PC when using different approaches.

participants are experienced in using PCs. Figure 10 shows that the delays with Bluetooth, LAN and Wi-Fi are mainly caused by the device pairing. Although MousePath has a larger transmission delay, the total delay is less.

VII. Discussion

Potential Extensions. MousePath implies interesting security properties worth further exploration. For example, MousePath can be used to establish a secure user input system for public computers. The user could use the keyboard of his/her personal smartphone to do the input for the application via MousePath. This is without worrying about the risk of input leakage even when the underlying computer system is fully compromised. The key challenge is how to establish a trusted data path between the smartphone and the application. We plan to combine trusted computing technologies with MousePath to achieve this goal.

Mouse Sensitivity. In practice, we found that there are optical mice that cannot respond to texture shifts in the smartphone's screen, and thus cannot be used in MousePath. This is because of many aspects, including the intensity of the backlight, the focal length of the mouse lens, the movement detection algorithms in DSP, *etc.* To use these mice, the user might need smartphones with a thinner and brighter screen. Another possible solution is on the mouse side. The optical mouse can actually report sampled raw images [22], thus the texture can be directly used for conveying data like a QR code. We plan to explore this property to improve the compatibility of MousePath in future work.

VIII. Conclusion

This paper presents MousePath, which is a lightweight communication channel between smartphones and computers. MousePath makes use of a smartphone's screen as the transmitter and an optical mouse as the receiver. It fills the gap of *device-to-web* information sharing. We envision that it can enable a lot of interesting web applications.

Acknowledgements

We sincerely thank our shepherd Stephan Sigg for his thoughtful guidance and constructive suggestions. We also thank anonymous PerCom reviewers for their valuable comments. This work is supported by the Key-Area Research and Development Program of Guangdong Province 2020B0101390001, the ShanghaiTech Startup Fund, the Shanghai Sailing Program 18YF1416700, the "Chen Guang" Program 17CG66 supported by Shanghai Education Development Foundation and Shanghai Municipal Education Commission, NSFC 62002224 and 62002150, and the Project of "FANet: PCL Future Greater-Bay Area Network Facilities for Large-scale Experiments and Applications" LZC0019.

References

[1] "Media Capture and Streams API," https://www.w3.org/TR/mediacapture-streams/, 2020.
[2] D. Antonioli, N. O. Tippenhauer, and K. Rasmussen, "Key negotiation downgrade attacks on bluetooth and bluetooth low energy," *ACM Transactions on Privacy and Security*, vol. 23, no. 3, pp. 1–28, 2020.
[3] F. Xu, W. Diao, Z. Li, J. Chen, and K. Zhang, "Badbluetooth: Breaking android security mechanisms via malicious bluetooth peripherals." in *NDSS*, 2019.
[4] W. Hu, G. Hao, and Q. Pu, "Lightsync:unsynchronized visual communication over screen-camera links," 2013.
[5] Z. Yang, Z. Wang, J. Zhang, C. Huang, and Q. Zhang, "Wearables can afford: Light-weight indoor positioning with visible light," in *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, 2015, pp. 317–330.
[6] B. A. Myers, "Using handhelds and pcs together," *Communications of The ACM*, vol. 44, no. 11, pp. 34–41, 2001.
[7] A. L. Simeone, J. Seifert, D. Schmidt, P. Holleis, E. Rukzio, and H. Gellersen, "A cross-device drag-and-drop technique," in *Proceedings of the 12th International Conference on Mobile and Ubiquitous Multimedia*, 2013.
[8] H. Xia, J. Zhang, Y. Zhu, C. Yu, and Y. Shi, "Mobile assistant: enhancing desktop interaction using mobile phone," in *ACM ITS*, 2012.
[9] A. Perrig, M. K. Reiter, and J. M. Mccune, "Reducing the trusted computing base for applications on commodity systems," 2009.
[10] M. Bachratý and M. Žalman, "2D Position Measurement with optical laser mouse sensor," Jan. 2010.
[11] AVAGOTECH, "Solid-State Optical Mouse Lens Data Sheet," https://www.sparkfun.com/datasheets/Widgets/AV02-1278EN.pdf, 2019.
[12] D. Tse and P. Viswanath, *Fundamentals of wireless communication*. Cambridge university press, 2005.
[13] P.-E. Forssén and E. Ringaby, "Rectifying rolling shutter video from hand-held devices," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, 2010, pp. 507–514.
[14] J. Chen and P. Owsley, "A burst-error-correcting algorithm for Reed-Solomon codes," *IEEE Transactions on Information Theory*, vol. 38, no. 6, pp. 1807–1812, Nov. 1992.
[15] "Eye Pattern," https://en.wikipedia.org/wiki/Eye_pattern, 2020.
[16] "Google Map API," https://developers.google.com/maps/documentation/geolocation/overview, 2020.
[17] "Android Wi-Fi Localization ," https://developer.android.com/guide/topics/connectivity/wifi-rtt, 2020.
[18] "KeePassDroid," www.keepassdroid.com, 2020.
[19] Y.-D. Lin, E. T.-H. Chu, E. Chang, and Y.-C. Lai, "Smoothed graphic user interaction on smartphones with motion prediction," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2017.
[20] "Mouse Lock API - Web APIs," https://developer.mozilla.org/en-US/docs/Web/API/Pointer_Lock_API, 2019.
[21] "ipinfo.io," https://ipinfo.io/, 2020.
[22] M. M. Silva, J. R. D. A. Nozela, M. J. Chaves, R. A. Braga, and H. Rabal, "Optical mouse acting as biospeckle sensor," *Optics Communications*, vol. 284, no. 7, pp. 1798–1802, 2011.