CapCam: Enabling Quick, Ad-Hoc, Position-Tracked Interactions Between Devices

Robert Xiao **Scott Hudson** **Chris Harrison**
Carnegie Mellon University, Human-Computer Interaction Institute
5000 Forbes Avenue, Pittsburgh, PA 15213
{brx, scott.hudson, chris.harrison}@cs.cmu.edu

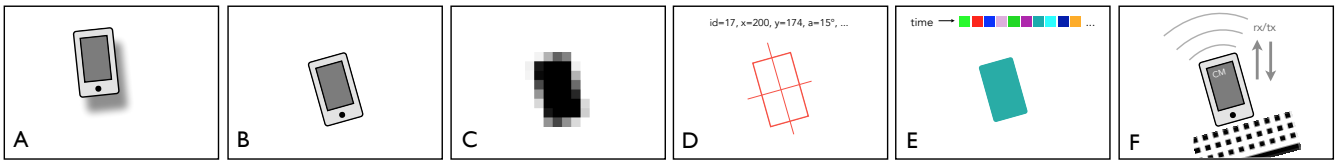

Figure 1. A: CapCam is used to pair two devices, a "cap" device (background is a large touchscreen display) and a "cam" device (here, a smartphone). B: The phone is pressed to the display. C: The phone body creates a characteristic signal on the touchscreen's capacitive sensor. D: CapCam extracts the shape, position and orientation of the phone from this capacitive image. E: CapCam encodes pairing data (e.g., IP, port and password) as a flashing color pattern, rendered beneath the phone body. F: The phone's rear camera captures the pattern, and uses it to establish a conventional two-way wireless link (e.g., WiFi). With both devices paired and communicating, interactive applications can be launched, such as this virtual keyboard.

ABSTRACT
We present CapCam, a novel technique that enables smartphones (and similar devices) to establish quick, ad-hoc connections with a host touchscreen device, simply by pressing a device to the screen's surface. Pairing data, used to bootstrap a conventional wireless connection, is transmitted optically to the phone's rear camera. This approach utilizes the near-ubiquitous rear camera on smart devices, making it applicable to a wide range of devices, both new and old. CapCam also tracks phones' physical positions on the host capacitive touchscreen without any instrumentation, enabling a wide range of targeted interactions. We quantify the communication performance of our pairing approach and demonstrate data transmission rates up to four times faster than prior camera-based techniques. To demonstrate the unique capability and utility of our system, we built a series of example applications, highlighting different interaction techniques CapCam enables.

Author Keywords
Large displays; ad-hoc connections; capacitive sensing; device pairing; cross-device interaction.

ACM Classification Keywords
H.5.2. Information interfaces and presentation (e.g., HCI): User interfaces: Input devices and strategies.

ISS '16, November 06-09, 2016, Niagara Falls, ON, Canada
© 2016 ACM. ISBN 978-1-4503-4248-3/16/11···$15.00
DOI: http://dx.doi.org/10.1145/2992154.2992182

INTRODUCTION
Large touchscreen displays, such as public kiosks, digital whiteboards and interactive tabletops have become increasingly popular as prices have fallen. Similarly, mobile devices, such as smartphones and tablets, have achieved ubiquity. While such devices are reasonably smart on their own, cross-device interactions hold much promise for making interactive experiences even more powerful [14].

However, interacting *across* devices is rarely straightforward. Although many mobile devices support e.g., Bluetooth pairing, such pairing options are generally time-consuming and cumbersome (i.e., on the order of 5 seconds, see Table 1). Often, users must confirm or manually enter connection parameters (e.g., device, network identifier) and security parameters (e.g., PIN) [27]. Short-range NFC, an emerging technology, aims to mitigate many of these issues. However, it requires specific hardware on both devices, and more importantly, only indicates device presence, not position (unless receivers are tiled into an matrix [35] or combined with another method, like optical fiducial tags [1]). Thus, it only allows for coarse device-to-device pairing, precluding metadata such as spatial position and rotation of devices, as well as rich multi-device experiences. Moreover, NFC is not commonly available on larger devices, such as laptops, tablets, and interactive surfaces – the class of surfaces we chiefly target.

CAPCAM
In this work, we describe *CapCam*, a new technique that provides rapid, ad-hoc connections between two devices. CapCam pairs a "cap" device with a capacitive touchscreen to a "cam" device with a camera sensor (Figure 1A). For example, typical smartphones and tablets can be paired with each other, and these devices can be paired to even larger touchscreens, such as smart whiteboards and

Method	Pair Time	Notes
Bluetooth	4.8 sec (SD 2.5)	Starting from Bluetooth config screen of smartphone with device already in discoverable mode. Mean of: Hyundai Elantra (2012), Samsung HM3500 Headset, Plantronics BackBeat 903 headset, Beats Studio Wireless headset.
Wifi WPS	5.1 sec (SD 1.3)	Starting in WiFi settings of smartphone, from moment of WPS button press. Mean of: LinkSys E3200 router, MyQ Garage door opener, Technicolor 582n router, D-Link N300 router.
NFC	1.3 sec (SD 0.3)	With item to send or NFC application already open, from time to tap devices together to time of visible response ("beam" prompt, notification, etc.). Mean of: Nexus 7, LG NFC Tag, WhizTag, Sony Xperia Z, Samsung Galaxy Nexus.
FlashLight [12]	2.64 sec (Tx only)	Pairing data transmission time alone, calculated from the paper. 100 ms camera synchronization time, plus 84-bit payload at 33 bits/sec (the highest throughput achievable without significant error). Data rate assumes error correction is not needed.
BlueTable [37]	3 sec (Tx only)	Pairing data transmission time alone, calculated using the statistics quoted in the paper. This system transmits 8-bit pairing codes at 2.67 bits/sec.
CapCam	1.21 sec (SD 0.1)	With app open, from time to tap Cam device to time of visible Cap device response (including network connection overhead). 84-bit message + 42 additional ECC bits. Time measured using worst-performing device (Microsoft Surface). **0.83** seconds to transmit the 126-bit pairing packet alone (for comparison with FlashLight and BlueTable). See also Video Figure.

Table 1. Informal comparison of different pairing methods.

touchscreen monitors. CapCam uses the cap device's touchscreen to detect and track the cam device (Figure 1, C and D), and renders color-modulated pairing data that is captured by the cam device's rear camera (Figure 1E).

This pairing data contains configuration information necessary to establish a bidirectional link (e.g., IP address, port and password). In this way, CapCam provides a unidirectional communication mechanism from the touchscreen to the camera, which is then used to bootstrap a full bidirectional, high-speed link (Figure 1F). Because CapCam also provides precise, continuous spatial tracking, we can enable rich synergistic applications utilizing both (or many) devices at once.

Overall, we believe CapCam exhibits six desirable properties – it enables *zero-configuration* pairing via automatically transmitted pairing codes; it is *rapid*, capable of establishing links in roughly one second; *anonymous*, in that it requires no identifying information to be exchanged; pairing is explicitly initiated by users through a *purposeful* pressing of a device to a host screen; it enables *targeted* interactions on said screen via position tracking; and, it allows for *multiple devices* to be paired and used on the same cap device simultaneously.

Although many prior systems have independently addressed pairing or spatial interaction, few have *combined* these into a single system. CapCam provides both pairing and spatial interaction as phases of a single interactive transaction, enabling rapid, ad hoc interactions, e.g,. walking up to a public display and initiating rich, spatial interactions nearly instantaneously.

In addition to describing our CapCam implementation, we also offer several applications and interactions enabled by our technique. We further contribute an evaluation of the technical aspects of our approach, including pairing latency, pairing code bandwidth and bit error rate across three exemplary devices.

RELATED WORK

CapCam intersects with several distinct areas of the literature, which we now review.

Device Pairing Interactions

Interaction techniques for establishing an ad hoc connection between devices in a convenient but secure fashion have been proposed in a large number of research systems. Chong et al. provide a survey of systems in [9] and review relevant usability factors in [8]. Systems differ in terms of the actions users must undertake to establish an association between devices: using e.g., pre-shared codes [20], simple placement of a phone in view of a camera on a specially augmented surface [37], gestures performed by the user with a mobile device [15, 25], or synchronous motion, vibration or other input on two devices [6, 13, 14, 15, 24, 28], among others.

Pairing Techniques using Optical Communication

A number of systems have explored optical transmission methods for device pairing [21, 26, 30, 31, 37]. Several use color encoding, e.g., [22] which uses hue differentials for coding; [38], which exploits the rolling-shutter effect to present imperceptible visual tags; and [12, 34], which use color transitions for coding. However, only one prior system, FlashLight [12] has demonstrated optical transmission of data between a display (a diffuse illumination touch table) and a coupled phone.

Augmenting Interactions Using Spatial Tracking

A phone or other device with knowledge of its spatial location can be used as an auxiliary input and/or display device to augment interaction on a larger device (see e.g., [17, 19, 23, 36]). For example, it could be used as a magic lens [2], providing physical detail+context [29], or as a *peephole* display [39] in a larger context. Other systems use specially-designed tangible widgets which can be tracked and identified by a capacitive touchscreen, e.g., TUIC [40] and CapStones and ZebraWidgets [7]. The phone's camera can also be used to infer the phone's position relative to a remote display, to enable various interactions [3, 4].

One exemplary system is PhoneTouch [32, 33], which lets users tap a phone onto a display surface and perform targeted interactions. This is achieved by merging two disparate events: an impact registered by the phone's accelerometer and the appearance of a new blob on an FTIR display.

This requires devices to be pre-paired and have synchronized clocks. Similarly, the Thaw system [18] uses a hue encoding technique to encode x-y positions on a display, which can be interpreted by the camera of a pre-paired phone. As previously mentioned, NFC pairs one device to another, without fine grain targeting. In response, Touch & Select [35] uses a special grid of NFC tags operating behind the display to provide rough position tracking of a phone while using NFC. Finally, Echtler et al. [10] use a camera-driven interactive table to recognize devices based on the shadows they cast, and combine this with Bluetooth signal strength to establish a connection, thus requiring no modification to e.g., smartphones.

Closely Related Systems

A few systems deserve special mention for their proximity to the present work. BlueTable [37] tracks phones on a surface using an overhead camera, and pairs with them via infrared flashes from the IrDA transceiver. BlueTable was able to transfer data at 2.67 bits per second using binary encoding. FlashLight [12] is a system that transmits pairing data from a diffuse illumination touch table to a phone resting on it; a basic color transition scheme is used to transmit approximately one bit per frame to the phone's camera, achieving a transmission rate of 33 bits per second.

In contrast to these systems, CapCam is the first to use commodity, off-the-shelf capacitive touchscreens, meaning our technique is *immediately portable* to most touchscreen devices today (as opposed to optical touch tables used in e.g., [10, 37]). Additionally, CapCam offers significantly higher data transmission rates – up to 150 bits per second – allowing for almost instant pairing. Table 1 offers findings from an informal comparison of commercial systems, BlueTable and FlashLight. Beyond offering competitive pairing ease and speed, CapCam also significantly extends prior work with respect to our exploration of the interactions enabled; please also see our Video Figure.

IMPLEMENTATION

The cap device in CapCam can be any capacitive touchscreen device, ranging from large public kiosks to small mobile phones. To demonstrate this range, we used several different off-the-shelf, consumer devices for development and testing. Importantly, these devices represent a wide range of sizes and pixel densities.

Smartphone – We used a stock Nexus 5 running Android 5.0.1, which features a 5" diagonal screen with a resolution of 1080x1920 pixels (445 pixels per inch) running at 60 fps, and a touchscreen update rate of 120 Hz.

Laptop – Laptops increasingly feature touchscreens, and so we chose a laptop as our second display platform. In order to directly compare transmission performance with FlashLight [12], we use a MacBook Pro. Because this particular computer lacks a touchscreen, we only evaluated data transmission rates with this device. The MacBook has a 15" screen with a resolution of 2880x1800 pixels (220 PPI) running at 60 fps.

Figure 2. A hand and smartphone resting on our Microsoft Surface display. The capacitive image is seen above (offset vertically for illustrative purposes).

Large Interactive Surface – As a demonstration of large CapCam-enabled interactions, we use a 55" Microsoft Surface touchscreen display, with a resolution of 1920x1080 pixels (40 PPI) at 60 Hz. The touchscreen has an update rate of 120 Hz.

Cam Device

As an example cam device, we used an unmodified Nexus 5 smartphone running Android 5.0.1. Android 5 provides low-level control over many parameters of the phone's camera, enabling specialized imaging applications like ours. Specifically, it allows full-manual control over shutter speed, exposure, focus and color compensation. Although few devices currently support this full-control API, the underlying features are present on nearly all image sensors at a low level (readily available to system integrators and hardware manufacturers).

Despite our control over the camera's focus parameters, the images are nonetheless blurred because the camera cannot focus at such short distances. We use the camera in manual-control video mode, which provides 640x480 resolution images at 30 FPS.

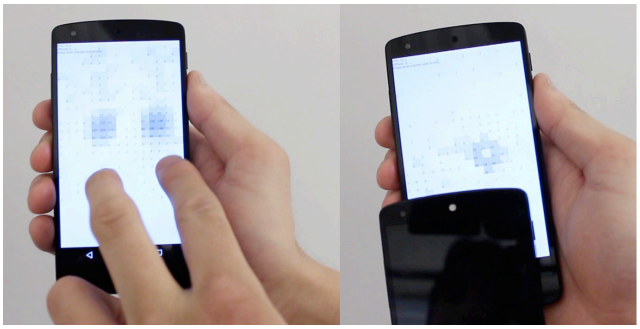

Figure 3. Capacitive images from our Nexus 5 smartphone showing the signal obtained from finger touches (left) and a phone contact (right). The camera ring of the contacting phone is visible at right. For illustrative purposes, the signal is offset vertically to avoid occlusion.

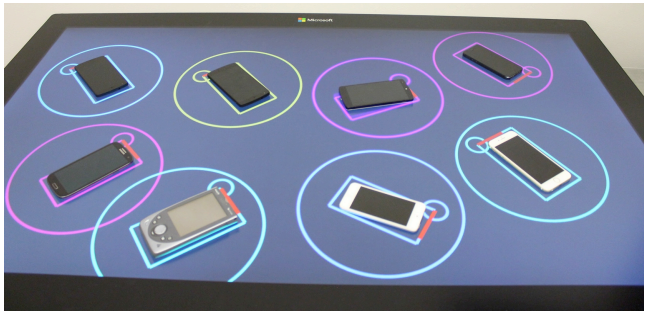

Figure 4. Multiple devices (of many models) simultaneously tracked on the Microsoft screen by our algorithm. Rectangles display the recognized phone's shape, orientation and position. A red line denotes a device's x-axis; the upper-left corner is marked with a hollow circle.

Capacitive Image Processing and Tracking

Capacitive touchscreens function by sensing changes in the electric field near the surface of the display. Typically, the field is sensed at points on a regularly-spaced electrode grid, resulting in a low-resolution "capacitive image". The touch controller processes this image internally to extract touch data (most typically, the X/Y position of touch contacts), which is then passed up to the operating system.

Most touch controllers, however, offer a "debug" interface that enables the complete capacitive image to be retrieved. On the Nexus 5 smartphone, we modified the Linux kernel to provide access to the Synaptics S3350 touch controller's capacitive image (a capability previously exploited in e.g., CapAuth [11] and BodyPrint [16]). On the Microsoft Surface display, we developed a user-space USB driver for OS X integrated into the cap application, which retrieves the image over the USB connection. We use the raw capacitive image (Figures 2 and 3) produced by the touchscreen controller to implement our own tracking algorithms, which we designed for finding phones and touches simultaneously (Figure 4 and Video Figure).

On our large interactive display, we find phones by tracking the rectangular capacitive imprint (Figure 2, right). When a phone is placed on the surface, the phone creates a characteristic electric field disturbance under the entire area of the phone. Although the capacitive measurement of the phone is generally lower than that of fingers, the noise level of the Microsoft Surface display precludes using a simple threshold approach. Instead, we use a connected components algorithm to detect blobs in the image. The blob size is used to determine whether the blob represents a phone or a finger touch. A natural consequence of this simple approach is that large touch contacts, such as palms, may also be recognized as "phones". However, since non-cam-device contacts will not pair with the screen, they can be rejected quickly.

For phones, we then fit a rectangle to the contact blob to extract the approximate bounding box of the phone (Figure 4). To obtain the phone's angle, the algorithm selects the orientation angle that minimizes the bounding box size. After the phone establishes a connection with the display, it can optionally transmit its physical dimensions to the display to enable more precise tracking.

On the Nexus 5 smartphone, the touchscreen controller's built-in auto-calibration feature, which integrates large, weak capacitive blobs into the background profile over time, precluded us from similarly tracking the body of the other phone. Consequently, we track other phones (*i.e.*, cam devices) by identifying and tracking the metallic ring that often surrounds camera modules (seen in Figure 3, right). We use a connected-components algorithm (flood fill algorithm) to locate the ring, and distinguish it from fingers by comparing the maximum signal value against a threshold – the capacitive measurement of the ring is significantly lower than any finger.

Of note, the lower resolution of our Surface's touchscreen prevents us from reliably segmenting the camera module from the rest of the phone's blob. Instead, we treat any contact point inside the phone's area as a potential camera. In order to accurately discern the camera's position, we vary the pairing data (e.g., using different passcodes) rendered beneath the phone by position (e.g., quadrants) – the unique pairing data that eventually connects thus reveals the camera location. As camera modules are almost always located at the top of devices, we can further infer the orientation of devices (Figure 4, top left corner denoted with a circle).

Transmitting the Pairing Data

We encode pairing data as visible light beneath the device. Pairing data is encoded into sequential frames, where each frame consists of a color. Thus, the pairing data appears visually as a series of flashing solid colors on the display (see Video Figure). To match the frame rate of the camera, the frames are transmitted at 30 Hz (thus, when displayed at 60 Hz we simply repeat each color frame twice).

Each packet of pairing data begins with a three-frame header, consisting of one frame each of solid green, red and blue (Figure 5). This sequence is highly unlikely to occur in the wild or at random, especially at precisely 30 Hz, and so it neatly delineates packets. Furthermore, as we describe later, this known and "pure-color" header data is used to automatically color calibrate the camera to the particular display for improved payload decoding.

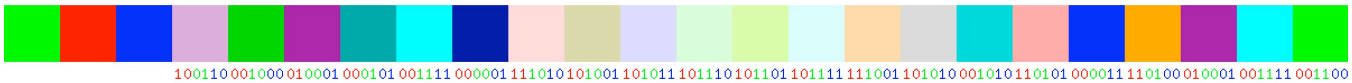

Figure 5. A sample packet as encoded by the display. The packet starts with a three-frame green-red-blue header, followed by 21 color frames containing 6 bits each (using 4 color levels per channel). The raw transmitted bits are noted below each frame.

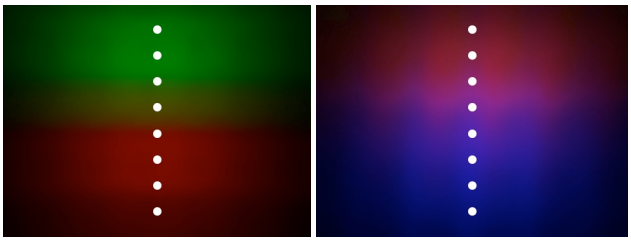

Figure 6. Two consecutive camera frames captured by our Nexus 5, showing our GRB packet header (Figure 5). Image rows are captured sequentially in time due to the rolling shutter. Left: the green header frame, followed by the start of the red frame. Right: the next frame shows the end of the red frame and the start of the blue frame. White dots represent the sampling points used to extract the packet.

Following the header, the pairing data is encoded into the RGB color values of each frame (Figure 5), depending on the number of color levels configured. For example, if we use 4 color levels per channel (e.g., color channel values of 0, 85, 170, and 255), then each color channel encodes 2 bits of data (00, 01, 10 and 11 respectively). Since we can independently manipulate R, G and B output on the display, each frame encodes 6 bits. Using this scheme, a 162-bit message can be transmitted in one second (30 frames total, three header frames plus 27 data frames). Increasing the number of levels per channel provides higher theoretical throughput, but also raises the error rate. Consequently, the number of levels should be chosen to optimize the error-corrected bit rate (Figure 9).

Modern operating systems perform sophisticated color correction to match display characteristics with human perception. These color corrections include gamma corrections, chromatic shifts, white-point adjustments, and gross brightness/contrast control. Although these are useful for presenting accurate colors to humans, they cause unpredictable changes in the displayed color data across monitors. We therefore bypass these color correction algorithms by specifying that our color data is pre-corrected to the sRGB color space, thus allowing us to display "raw" RGB values.

As LCD displays vary in emission spectra and brightness, there will still be a color mismatch between the display and the camera. We use our GRB packet header (Figures 5 and 6) to calibrate the camera, computing a color-calibration matrix that reduces the observed cross-talk between different channels and normalizes the sensitivity of the camera's response to each channel. After this correction, there is still some residual noise, for example, some LCD displays exhibit slow switching times, resulting in intermediate color values being received by the camera. Because of this, some level of error correction is required to accurately transmit packets. Furthermore, these effects may reduce the number of usable color levels per channel.

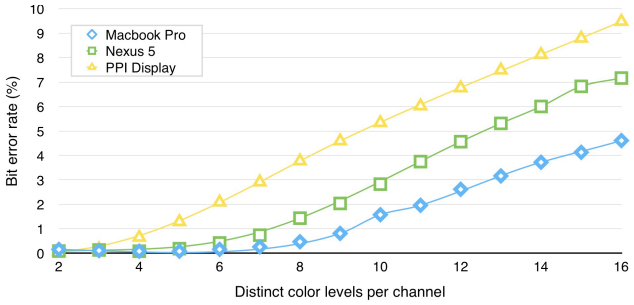

Figure 7. Raw bit error rate (% of bits flipped) as a function of color level density.

Capturing the Connection String with the Camera

Our example displays all have a refresh rate of 60 Hz, accurate to within fractions of a Hz. Due to the rolling shutter on the phone's camera (true for most cameras used in smartphones), the camera frame will usually contain part of one display frame, the vertical blanking interval, and part of the next display frame (Figure 6). The point in the image at which the blanking interval appears will move as the camera moves in and out of sync with the display. If the break moves across the image too quickly, the captured frames will skip or lag with respect to the display, invalidating the received data.

Thus, we configure the phone's shutter speed so that it captures frames as close to 30 Hz as possible, by setting the sensor frame duration as close to 33.3 ms as the hardware will allow. On the Nexus 5, we can do so to within 0.3 Hz (33.1 ms), so that the blanking interval moves at most 1/3 of the camera frame per second. This is sufficiently stable for our use. We used the same camera settings for all smartphones and cap devices.

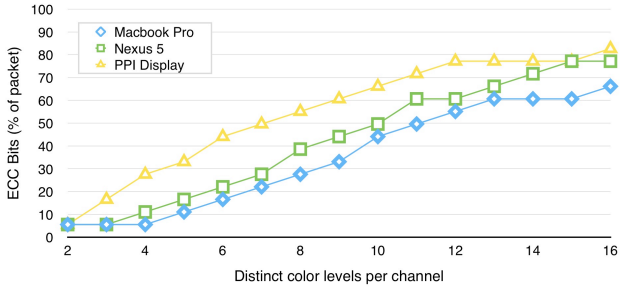

Figure 8. Percentage of 126-bit packet needed for error correction to attain a packet loss rate of 1%, as a function of color level density.

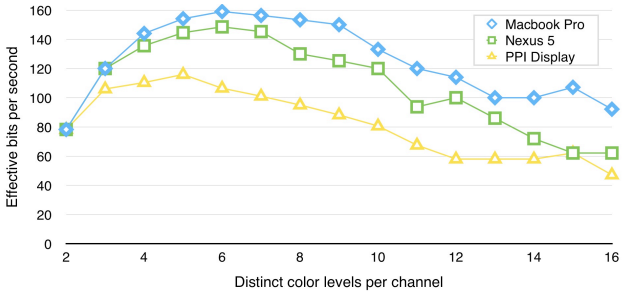

Figure 9. Effective payload data transmission rate as a function of color level density, after applying the error correction necessary for a target packet loss rate of 1%.

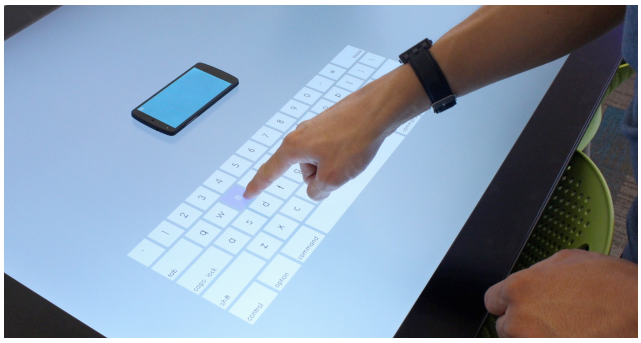

Figure 10. Keyboard example. The screen provides a large on-screen keyboard for paired phones, easing text input.

On the cam device, we select eight sample points across the height of the camera frame (Figure 6, white dots). At each sample point, we attempt to decode the observed color sequence as a packet. Because we synchronize closely to the display's frame rate, there will be at least one sample point that is not affected by the rolling shutter break for the duration of the transmission, allowing that sample point to receive the message and decode it successfully.

Connection Creation

Once the cam device receives and decodes the pairing data packet, it initiates a connection to the display. In our proof-of-concept implementation, the connection is made over WiFi, though any wireless protocol could be used (e.g., cellular). The pairing data contains all data necessary to establish this connection. For example, it could contain a Bluetooth hardware address and PIN, a WiFi ad-hoc BSSID and WPA2 key, or an IP address and port if the devices are on a shared network (e.g., cellular network or WiFi access point). The pairing data packet also contains a one-time use passcode (i.e., nonce), which further identifies the cam device to the cap device.

Error Correction

Due to inherent noise and non-uniformity in the camera hardware, color space differences, and LCD imperfections, some level of error correction is needed to ensure reliable data transmission. In our implementation, we employ a BCH error-correcting code [5] appended to each transmitted packet. The precise code configuration used can be optimized based on the capabilities of the screen; for our implementation, we used a 42-bit code appended to an 84-bit message, capable of correcting any 6 erroneous bits within a 126-bit packet. These parameters were chosen to give a 99% packet transmission rate on the Microsoft Surface device (which exhibited the most challenging transmission characteristics, as seen in Figure 7).

Connection Tear Down

When the cam device is removed from the display, the cap device immediately detects the loss of capacitive tracking and can optionally terminate the connection if no further interaction is desired. This facilitates rapid and effortless creation and destruction of ephemeral pairings between the devices. Moreover, this serves to require physical proximity for pairings to exist, which is not possible with a purely wireless solution. Alternatively, applications can choose to retain the pairing after the phone leaves the display, using CapCam as a general-purpose device pairing solution.

Privacy

Although CapCam was not designed for security-sensitive applications, it does exhibit several properties that make it reasonably robust against simple attacks (e.g., compared to that of entering a PIN into an ATM).

A co-located attacker with line of sight could conceivably intercept the data sent from the display to the phone. This line of sight is significantly hindered by the fact we render our visible light handshake under the device, thus requiring a camera to be at an oblique angle relative to the display and with a large depth of field. We could further obfuscate data transmission with false patterns presented around the camera module.

Even if the attacker is successful in visually intercepting the connection data, they would only gain access to the host display (not the client) and then, only for an extremely short period of time. Only one connection is permitted per port/passcode. Thus, the attacker would have to decode the CapCam packet before the phone; when the true client phone does attempt to connect (perhaps a few tens of milliseconds later), the server can terminate both connections on grounds of suspicious activity.

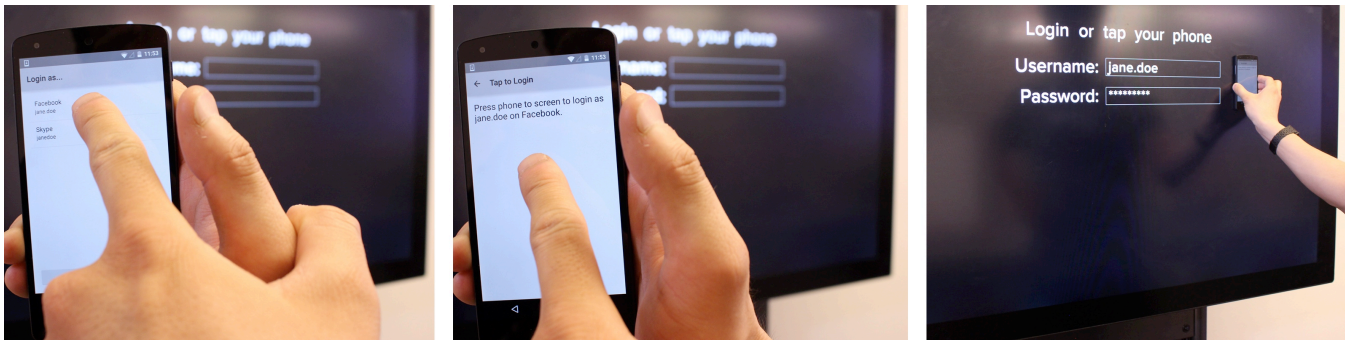

Figure 11. Authentication demo. Instead of entering a password on a highly-visible on-screen keyboard, users select an account (left) on their phone. The credentials are automatically sent after pressing the phone to the screen (right).

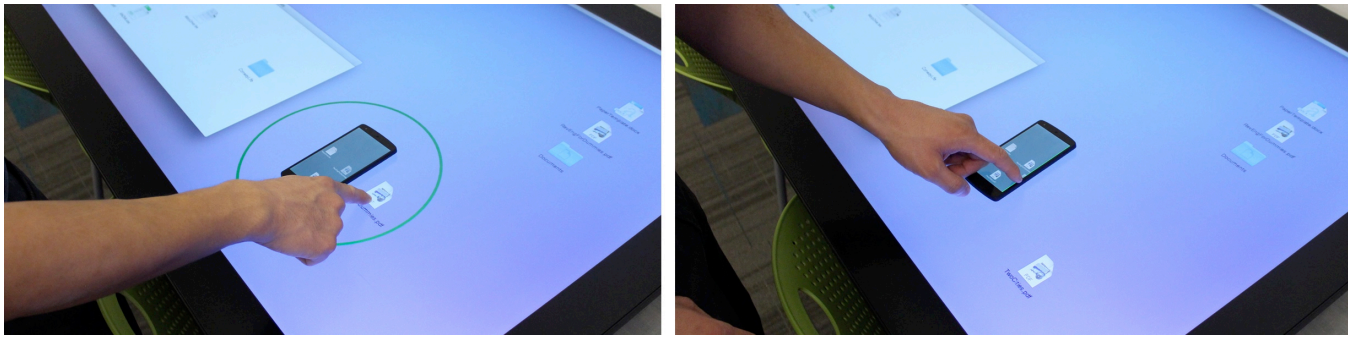

Figure 12. Data transfer between a phone and a desktop. Left: desktop files can be dropped onto the phone. Right: dragging files to the bezel of the phone drops them on the desktop.

In general, after successfully establishing a two-way wireless connection between legitimate devices, the host/client can negotiate an encrypted channel using e.g., Transport Layer Security (TLS) to thwart passive listening attacks.

EVALUATION

We conducted an evaluation to determine the transmission characteristics of CapCam: the bit error rate, packet drop rate, and effective transmission rates over a range of different transmission encodings and devices. Our evaluation was performed on our three test devices: a Nexus 5 smartphone, a MacBook Pro laptop, and a Microsoft Surface 55" multitouch display. A Nexus 5 smartphone was used in all three cases to receive the data.

Each of the host displays was configured to repeatedly flash a random data packet encoded with a random number of color levels (from 2-16 levels per channel) and a random packet length (from 10-20 display frames per packet), with no error correction applied. In each case, the packet was preceded and followed by the standard G,R,B header. After the display flashes the packet, the phone sends back the received packet (or reports a failure if the packet was not detected) and the result is compared against the original sent packet to determine the bit error rate.

RESULTS AND PERFORMANCE

Each display device was tested over a period of eight hours, during which the smartphone was placed on the cap device and configured to continually receive pairing data. In total, 113,957 packets were sent (over 32,000 per device); 64 packets were lost, for a packet loss rate of less than 0.06%. In total, 5,106,848 bits of information were transmitted.

For each device and number of color levels, we computed the total raw bit error rate, shown in Figure 7. The error rate increases as the number of color levels increases, due to the decreasing separation between adjacent color levels. We then calculated the minimum amount of error correction needed to ensure that 99% of packets are corrected (a packet loss rate of 1%) for each encoding level; the error correction levels are shown in Figure 8 for a packet size of 126 bits. In all cases, a BCH error correction scheme was used.

Finally, based on the resulting effective payload sizes, we computed the net throughput of the system, shown in Figure 9. The results show that there is a consistent "peak" of throughput around 4-6 color levels, past which the increased raw throughput is outweighed by the increase in error rate.

As our implementation is designed for a power-of-two number of color levels (for simplicity), we chose 8 color levels for the MacBook and 4 color levels for the Nexus 5 and Microsoft Surface implementations, for bit rates of 150, 135 and 110 bits per second, respectively. These rates are sufficient to send a full setup packet (32-bit IP address, 16-bit port, 32-bit passcode) in under one second on all devices. For higher-security applications, 1.5 seconds would allow for a 128-bit key to be transmitted.

Overall, our encoding scheme, coupled with our greater control over the camera's shutter timing and color correction, allows us to achieve speeds much higher than the color transition approach shown in e.g., [12,34].

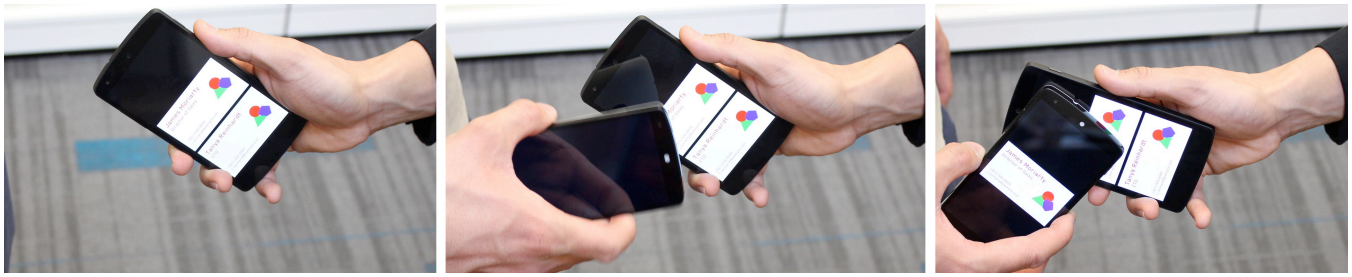

Figure 13. Business card exchange. After a meeting, electronic business cards are exchanged. Simply pressing one phone's camera to the other's screen initiates CapCam pairing and downloads the requested card.

Figure 14. Image gallery example. Users simply press their phone to a desired image to download it.

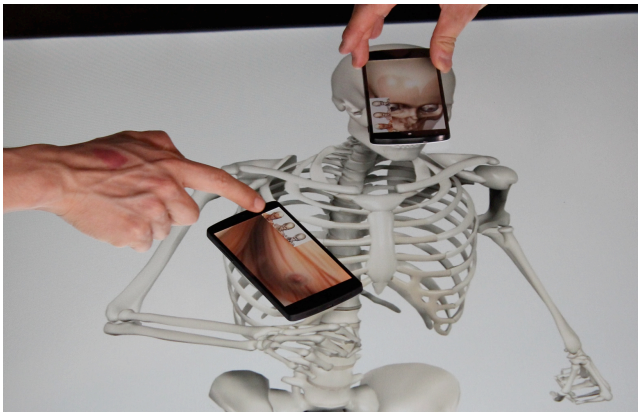

Figure 15. Context+Focus demo. The phones display high-resolution imagery that corresponds to the devices' position on screen. Users can also switch anatomical layers and change zoom levels.

EXAMPLE APPLICATIONS AND INTERACTIONS

To illustrate the potential of CapCam, we built seven example applications demonstrating different use cases. Please also see our Video Figure, which underscores the speed and robustness of our approach – *all* demos shown are functional and real-time.

Facilitating Input

We built a simple application that allows the user to enter text on their phone using a larger, on-screen keyboard (Figure 10). The user places their phone on a cap device (oriented like a table), causing a position-tracked keyboard to appear below the phone. Multiple devices can be paired to the same cap device, each given their own keyboard, which transmits keystrokes only to its respective device.

We also built a demo in which the phone transmits data to the host cap device. With public kiosks, users often have to enter their authentication credentials on a large touch-screen, where they can be easily observed. In this example, users can press their phone to the display to automatically transmit their credentials rapidly and securely (Figure 11).

File & Information Transfer

Data transfer between a phone and a computer is often challenging. In our desktop demo application, users simply press their phone to the display to enable bidirectional drag-and-drop file transfer. Dragging files off the phone causes them to be copied to the desktop (Figure 12, right), and vice-versa (Figure 12, left). The cap device tracks the location of the phone, allowing its physical location to serve as a drop target. Additionally, we stream file icons and relative radial position to render a live preview of the drop location.

Alternately, users can select files to download by pressing the phone directly onto documents. In our gallery demo, users can download a high-res image simply by placing their phone camera to it (Figure 14). CapCam transmits a URL for the desired content to the phone, which then downloads and displays the image. We note in this example that, from the user's perspective, the act of pairing the devices has disappeared into the main interaction itself.

The cap device need not always be a large display. We created a phone-to-phone business card demo, where two users can exchange business cards by pressing their phones together (Figure 13). CapCam transmits an appropriate pairing code (e.g., Bluetooth device ID and PIN) and automatically downloads the selected card file.

Phone as Accessory

The phone can also be used as an accessory for larger displays. As an example, we created a context+focus demo where the phone (445 ppi) is used as a focus display for the larger, low-resolution (~40 ppi) cap device (Figure 15). The phone's position and orientation is precisely tracked and transmitted from the cap device to the phone, allowing a seamless overlay of shared content.

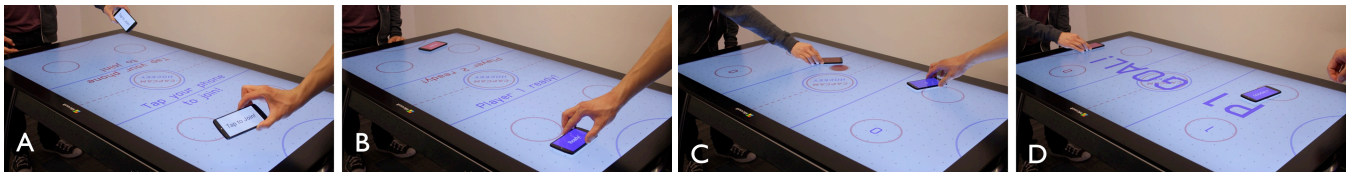

Figure 16. CapCam air hockey. (A) Players are invited to join the game. (B) When players press their phones to the table, CapCam rapidly and anonymously pairs the phones to the display. When two phones are paired, the game begins. (C) Players use their physical phones to deflect the virtual puck. CapCam tracks the phones and their orientations on the screen. (D) Sounds, vibrations and player-specific information appear on the phone, directed by the game through the paired connection.

Finally, we developed an air-hockey game using CapCam. In this demo, two users pair their phones to the cap device to start a match. The phones are used as physical paddles to hit the virtual puck (Figure 16). Furthermore, the phones provide stereo audio feedback (e.g., puck hit sounds, goal sirens, and ambient audience sounds), and even individualized haptic feedback (e.g., short vibrations when the puck is hit, long vibrations when a goal is scored).

DISCUSSION

Having been the focus of more than a decade of research and development, NFC is a mature technology offering a useful gold standard. Perhaps the most important factor is pairing time, in which CapCam and NFC are comparable. However, compared with NFC-based approaches, CapCam inherently provides targeted and position-tracked interactions. With spatial differentiation, multi-device (and multi-person) interactions are readily supported, opening new and different interactive opportunities. Further, CapCam requires physical and direct contact to pair, as opposed to NFC, which can pair at a distance of up to 10 cm without line-of-sight (e.g., accidently through a pocket).

Although CapCam transmits data much slower than NFC (150 bits/sec vs. 424 kbits/sec), the main goal of CapCam's pairing phase is simply to establish a unique, secure, high-speed bidirectional link using wireless communication protocols already present on most devices, much like NFC does when attempting to transfer files. There are also avenues to boost CapCam's transmission rate, for example by transmitting different data across the pixels beneath the camera to achieve spatial modulation.

CapCam presently requires a special application to be downloaded to a phone before pairing can proceed. Thus, the first time a user wishes to pair using CapCam, an application would have to be downloaded and run, adding to the pairing time. This, of course, is a one-time cost. Moreover, there is no reason that CapCam could not be integrated into the OS as a system level feature, just like NFC on many phones. Thus, we see this limitation chiefly as an artifact of prototyping and not an inherent quality of the technique. Finally, due to the use of the rear camera, power usage is fairly high during the CapCam pairing process. This could be mitigated with custom hardware, such as a set of color photodiodes connected to a low-power dedicated decoder circuit, enabling always-on code detection.

CONCLUSION

We have presented CapCam, a technique that allows two devices to establish rapid and ad-hoc connections simply by pressing the camera of one device to the touchscreen of another. By exploiting capacitive sensing, CapCam tracks the precise position of the camera device, which combines with rapid pairing to enable a wide range of interactive functionality. From a technical perspective, we showed that CapCam can transfer data from the screen to the camera over 100 bits per second with negligible error, several times faster than prior display-to-camera transmission schemes. We use this to transmit data needed to establish a quicker and more stable connection. Finally, we demonstrated a wide range of fun and useful applications that showcase the capabilities of CapCam.

ACKNOWLEDGEMENTS

This research was generously supported by the David and Lucile Packard Foundation, a Google Faculty Research Award and Qualcomm.

REFERENCES

1. Bazo, A. and Echtler, F. Phone proxies: effortless content sharing between smartphones and interactive surfaces. In *Proc. EICS '14*. 229-234.
2. Bier, E., Stone, M., Pier, K., Buxton, W. and DeRose, T. Toolglass and magic lenses: the see-through interface. In *Proc. SIGGRAPH '93*. 73-80.
3. Boring, S., Altendorfer, M., Broll, G., Hilliges, O. and Butz, A. Shoot & copy: phonecam-based information transfer from public displays onto mobile phones. In *Proc. CHI '07*. 24-31.
4. Boring, S., Baur, D., Butz, A., Gustafson, S. and Baudisch, P. Touch projector: mobile interaction through video. In *Proc. CHI '10*. 2287-2296.
5. Bose, R.C. and Ray-Chaudhuri, D.K. On A Class of Error Correcting Binary Group Codes. Information and Control, 3(1), March 1960. 68–79.
6. Castelluccia C. and Mutaf, P. Shake them up!: A movement-based pairing protocol for CPU constrained devices. In *Proc. MobiSys '05*. 51–64.
7. Chan, L., Müller, S., Roudaut, A. and Baudisch, P. CapStones and ZebraWidgets: sensing stacks of building blocks, dials and sliders on capacitive touch screens. In *Proc. CHI '12*. 2189-2192.
8. Chong M. and Gellersen, H. Usability classification for spontaneous device association. *Pers. Ubi. Comput.* 16, 1 (Jan. 2012), 77–89.
9. Chong, M., Mayrhofer, R. and Gellersen, H. A Survey of User Interaction for Spontaneous Device Association. *ACM Comput. Surv.* 47, 1, Article 8 (May 2014), 40.
10. Echtler, F., Nestler, S., Dippon, A. and Klinker, G. Supporting casual interactions between board games on public tabletop displays and mobile devices. *Personal Ubiquitous Comput.* 13, 8 (November 2009), 609-617.
11. Guo, A., Xiao, R. and Harrison, C. CapAuth: Identifying and Differentiating User Handprints on Commodity Capacitive Touchscreens. In *Proc. ITS '15*. 59-62.
12. Hesselmann, T., Henze, N. and Boll, S. FlashLight: optical communication between mobile phones and interactive tabletops. In *Proc. ITS '10*. 135-138.
13. Hinckley, K. Synchronous gestures for multiple persons and computers. In *Proc. UIST '03*. 149–158.

14. Hinckley, K., Ramos, G., Guimbretiere, F., Baudisch, P. and Smith, M. Stitching: pen gestures that span multiple displays. In *Proc. AVI '04*. 23-31.

15. Holmquist, L.E., Mattern, F., Schiele, B., Alahuhta, P., Beigl, M. and Gellersen, H. Smart-Its Friends: A Technique for Users to Easily Establish Connections between Smart Artefacts. In *Proc. UbiComp '01*. 116-122.

16. Holz, C., Buthpitiya, S. and Knaust, M. Bodyprint: Biometric User Identification on Mobile Devices Using the Capacitive Touchscreen to Scan Body Parts. In *Proc. CHI '15*. 3011-3014.

17. Lakatos, D., Blackshaw, M., Olwal, A., Barryte, Z., Perlin, K. and Ishii, H. T(ether): spatially-aware handhelds, gestures and proprioception for multi-user 3D modeling and animation. In *Proc. SUI '14*. 90-93.

18. Leigh, S.-W., Schoessler, P., Heibeck, F., Maes, P. and Ishii, H. THAW: tangible interaction with see-through augmentation for smartphones on computer screens. In *Proc. TEI '15*. 55-56.

19. Li, F., Dearman, D., and Truong, K. Virtual shelves: interactions with orientation aware devices. In *Proc. UIST '09*. 125-128.

20. Lin, F. X., Ashbrook, D. and White, S. RhythmLink: securely pairing I/O-constrained devices by tapping. In *Proc. UIST '11*. 263-272.

21. McCune, J., Perrig, A. and Reiter, M. 2005. Seeing-is-believing: Using camera phones for human-verifiable authentication. In *Proc. IEEE S&P '05*. 110–124.

22. Miyaoku, K., Higashino, S. and Tonomura, Y. C-blink: a hue-difference-based light signal marker for large screen interaction via any mobile terminal. *Proc. UIST '04*. 147-156.

23. Olwal, A. LightSense: enabling spatially aware handheld interaction devices. In *Proc. ISMAR '06*. 119-122.

24. Park, D.G., Kim, J.K., Sung, J.B., Hwang, J.H., Hyung, C.H. and Kang, S.W. TAP: touch-and-play. In *Proc. CHI '06*. 677-680.

25. Patel, S., Pierce, S. and Abowd, G. A gesture-based authentication scheme for untrusted public terminals. In *Proc. UIST '04*, 157-160.

26. QR Code. Automatic identification and data capture techniques. Bar code symbology. BS ISO/IEC 18004:200.

27. Rekimoto, J. Pick-and-drop: a direct manipulation technique for multiple computer environments. In *Proc. UIST '97*. 31-39.

28. Rekimoto, J. SyncTap: Synchronous user operation for spontaneous network connection. *Pers. Ubi. Comput. 8, 2* (May 2004), 126–134.

29. Rohs, M. and Oulasvirta, A. Target acquisition with camera phones when used as magic lenses. In *Proc. CHI '08.* 1409-1418.

30. Roth, V., Schmidt, P. and Guldenring, B. The IR ring: authenticating users' touches on a multi-touch display. In *Proc. UIST '10*. 259-262.

31. Saxena, N., Ekberg, J., Kostiainen, K. and Asokan, N. 2006. Secure device pairing based on a visual channel. In *Proc. IEEE S&P '06*. 306–313.

32. Schmidt, D., Chehimi, F., Rukzio, E. and Gellersen, H. PhoneTouch: a technique for direct phone interaction on surfaces. In *Proc. UIST '10*. 13-16.

33. Schmidt, D., Seifert, J., Rukzio, E. and Gellersen, H. A cross-device interaction style for mobiles and surfaces. In *Proc. DIS '12*. 318-327.

34. Schwarz, J., Klionsky, D., Harrison, C., Dietz, P. and Wilson, A. Phone as a pixel: enabling ad-hoc, large-scale displays using mobile devices. In *Proc. CHI '12*. 2235-2238.

35. Seewoonauth, K., Rukzio, E., Hardy, R. and Holleis, P. Touch & connect and touch & select: interacting with a computer by touching it with a mobile phone. In *Proc. MobileHCI '09.* Article 36 , 9 pages.

36. Wang, J., Zhai, S. and Canny, J. Camera phone based motion sensing: interaction techniques, applications and performance study. In *Proc. UIST '06*. 101-110.

37. Wilson, A.D. and Sarin, R. BlueTable: connecting wireless mobile devices on interactive surfaces using vision-based handshaking. In *Proc. GI '07*. 119-125.

38. Woo, G., Lippman, A. and Raskar, R. VRCodes: Unobtrusive and active visual codes for interaction by exploiting rolling shutter. In *Proc. ISMAR '12*. 59-64.

39. Yee, K. Peephole displays: pen interaction on spatially aware handheld computers. In *Proc. CHI '03*. 1-8.

40. Yu, N., Chan, L., Lau, S. Y., Tsai, S., Hsiao, I., Tsai, D., Hsiao, F., Cheng, L., Chen, M., Huang, P. and Hung, Y. TUIC: enabling tangible interaction on capacitive multi-touch displays. In *Proc. CHI '11*. 2995-3004.