Micro-Interactions with NFC-Enabled Mobile Phones

Ben Dodson Monica S. Lam

Computer Science Department
Stanford University
CA 94305
{bjdodson,lam}@cs.stanford.edu

Abstract. This paper coins the term *micro-interactions* to refer to the class of small exchanges between devices that occur almost instantaneously. For example, a mobile payment using near-field communication (NFC) is a micro-interaction. The arrival of NFC on smart phones makes possible a wide array of applications using micro-interactions, from sharing photos between a phone and a TV to checking a car into a valet parking service by touching two phones.

This paper addresses the challenge of how to create intuitive, frictionless micro-interactions that require no pre-configuration for a large class of applications. We deliver a consistent *tap-and-share* interface for many forms of micro-interactions through several concepts. We propose *interaction manifests* as universal descriptors of multi-party, cross-platform applications. Zero-click overheads are made possible by automatically using the foreground application as the context for the micro-interactions. We extend the concept of *connection handovers* to allow NFC-enabled applications to run unmodified on devices lacking NFC. We also show how these abstractions make it easy to create a variety of applications. All the application and library code is available as open source.

We demonstrate that by focusing on micro-interactions, our mobile phones can provide a single focal point that enables sharing of our digital identity, assets, applications, and personality with friends (with their mobile phones) as well as the larger-screen PCs and TVs all around us.

1 INTRODUCTION

The smart phone, being powerful, personal, always with us, always-online, is changing our everyday life. It will eventually hold the key to our identities, access rights to digital assets, personal communications, photos, media, etc. In a sense, the smart phone is an extension of our digital self. As such, there are many applications where we wish to share our digital personality on the phone with other people's phones around us, as well as with our surrounding devices. In many cases, we are sharing very small amounts of information, such as a phone number. People would not bother automating such interactions unless the overhead is kept to a minimum. We refer to such interactions as *micro-interactions*; they will not be used unless they are as frictionless as micropayments.

1.1 Near-Field Communication

Micropayment is poised to be widely enabled on the smart phone. It is implemented using Near-Field Communication (NFC). NFC is a radio technology that supports transactions at distances of a few centimeters. During a transaction, one party can be completely inactive, drawing power inductively from the active party. Even the active party draws little power and can be left on all the time with minimal effect on the phone's overall power draw. Also, the nearness of NFC transactions creates the possibility of using proximity as context and triggering an appropriate action almost instantaneously.

NFC, in the form factor of a credit card, has been used widely in Japan, Hong Kong, and other parts of the world for many years: for public transportation, vending machines, and convenience stores. Standards have also been created for "smart posters" [5]; posters, signs, and magazine pages can possess cheap, embedded data tags that contain information such as details of museum exhibits, transportation schedules, discount coupons, movie clips, or links to e-commerce sites. A third important use of NFC is for making long-lasting connections between electronic devices—simply touching the devices together will configure them to connect over a longer-range protocol such as Bluetooth or Wi-Fi.

Availability of NFC on smart phones presents an exciting opportunity. The ubiquity of mobile phones means that most consumers in the future will have access to this technology. The programmability means that many applications can be developed to handle the context of an NFC interaction. NFC allows our phones to easily communicate directly, without requiring a third-party server. The effortless connection of NFC opens up many opportunities for phones to enhance our physical social encounters.

1.2 Micro-Interactions on the Phone

The concept of device-to-device micro-interactions has been explored by many researchers [10, 14, 19, 26]. This paper explores specifically the new opportunities of micro-interactions as offered by the smart phone. How do we *frictionlessly* share the digital personality we have acquired on the phone with devices around us? How do we *frictionlessly* invite others to participate in applications we are running currently on our phone?

Consider sending a text message from a mobile phone. We may find ourselves near computers with keyboards and wish we could type on the keyboard instead of on the phone. Writing an SMS is a short-lived task, and so any setup (such as Bluetooth pairing) or added complexity will affect our willingness to adopt such a workflow. The tear-down must also be trivial— we do not want to permanently pair our PC's keyboard with the phone.

As an example, we have developed an application called TapBoard based on the ideas we present in this paper. With TapBoard, the user simply touches the phone with an NFC tag on the keyboard of a PC (a micro-interaction), the PC will bring up a webpage with a simple text box, as shown in Fig. 1. Any text typed in the text box shows up on the phone instantaneously. Closing

the webpage disassociates the phone from the PC. The NFC tag on the PC simply contains the information that enables the phone to connect to the PC, e.g. through Bluetooth, without user intervention.

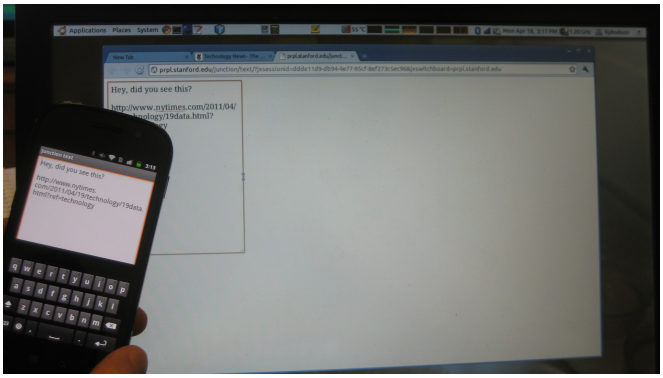

Fig. 1. Using TapBoard to enter mobile text.

Critically, the interaction occurs without requiring even a keypress. The UI is presented on the PC without the user searching for an application to launch, and the application does not hijack the user's PC usage in a permanent way. Even slightly increasing the difficulty of running the application such as accessing it from the application list or delays in loading the program may result in an experience that doesn't justify the gain in usability.

1.3 Contributions

This paper makes the following contributions:

1. We present a large number of scenarios where micro-interactions on the phone can be used in our daily life. We show that there are three major kinds of peer-to-peer micro-interactions: phone to phone, phone to another interactive device like a PC, phone to a passive device like a TV.
2. While inspired by NFC, the usage of micro-interactions we envision goes well beyond what can be implemented directly with NFC. We are able to deliver a consistent "tap and share" interface for many forms of micro-interactions with several novel concepts. *Interaction manifests* and the *Junction application platform* enable the sharing of not just data but decentralized multi-party applications across different platforms. Zero-click overheads are made possible by automatically using the foreground application as the context for the micro-interaction. Finally, *connection handovers* allow sharing across devices without NFC radios and for supporting continued interactions beyond the first touch.
3. All the abstractions presented in this paper are embodied in the publicly available Junction application framework and several libraries (EasyNFC,

LegacyNFC, and DesktopNFC). We have written a wide collection of applications (data sharing, keyboard sharing, remote presentation, and a multiparty poker game) using the libraries. Our experience suggests that the abstractions are powerful in simplifying the development of micro-interactions.

2 USES OF P2P MICRO-INTERACTIONS

Although micro-interactions are simplifying only small tasks, their wide applicability can have a major impact on our daily life. We imagine in the future, a child will instinctively touch his phone with different devices if he wants to share whatever he is doing to that device. Let us first describe some everyday scenarios with micro-interactions, then show how they can be organized as three major use cases.

2.1 Scenarios

We illustrate with scenarios below how pervasive micro-interactions can become, at home, in the work place, and when we are out and about. Several of the following scenarios are reminiscent of Mark Weiser's vision of ubiquitous computing, made possible by the NFC-equipped smart phones of today [25].

At home. Consider the mundane task of turning on the alarm on the phone before turning in every night. With micro-interactions, we can simply put the phone on our bedside table and it enters into "night mode," silencing our non-critical notifications such as when we receive emails.

A digital photo frame can now be set by simply touching the frame with a phone. If the phone is showing a picture, that picture gets on the frame; if the phone is showing a contact, our contact gets permission to remotely set photos on that frame.

In the living room, our phone turns into a digital remote by touching the phone to the remote. We can then change channels with the phone by browsing or searching or even using voice control. We can also browse for multimedia files on our phone and send them to the TV by touching the remote again, and use our online services like Netflix and Amazon to stream purchased content to our TV.

We can track our workout routine on our phone, to see a graphical representation of our progress. Bringing our phone near a scale picks up our weight and other vitals (kept privately on our device!), and touching our phone to our sports watch transfers the duration of our last run.

At the office. On our way to the office, we touch our phone to our car's center console to personalize the driving experience. It synchronizes our downloaded music and favorite radio stations and adjusts the car seat to our preference. We can also set our navigation device's destination by opening our phone's appointment book and again touching it to the console.

Suppose next we visit a company. To get a guest badge, we open the invitation from our host on our phone and touch it to the kiosk at the receptionist's desk to share who we are and with whom we are meeting. As we wait to meet our host, an email comes in that requires a lengthy reply. There is a guest computer nearby, and we touch our phone to it to borrow its keyboard.

If we are giving a presentation at a meeting, we touch our phone to the conference room's projector to bring up our presentation. The phone acts as a controller for the talk. Touching our phone to other computers in the room brings up the presentation on those devices as well. We can also touch our phone to attendees' phones in the room to give them a copy of our slides.

Out and about. Suppose we want to use valet parking at a restaurant. We hand over the keys to our car, and touch a kiosk to get a digital version of our valet ticket. We send the kiosk a picture of our face so they can recognize us when we return. We "check in" at the restaurant by touching our phone to the hostess's station. When we're done eating, we pay for the lunch using our phone, and the receipt is automatically stored in our device. Our phone knows if we are on a business trip, and automatically forwards the receipt for reimbursement. Heading back, we say "get the car" to our phone, and the valet is notified that we're on our way back, so the car is ready when we return.

Suppose next we wish to go to a party on public transportation. We touch our phone to a sign at the bus stop to get the schedule of when the next bus is coming. If the wait is too long, one click lets us call a cab. At the party, we touch our phone to a sticker at the door to check in. We can see a list of everyone else who's checked in and get their contact information. There's also a TV playing music and showing photos. Because we've checked in, we can choose music to play from our personal collection. And until we "check out", the photos we capture are sent to the running slideshow instantly.

2.2 Kinds of Micro-interactions

All the micro-interactions described above can be categorized according to the relationships of the interacting parties.

Multi-party (e.g. phone to phone). The interacting devices belong to different individuals; either or both of the parties may wish to initiate an interaction. A user may want to share the document or application he is viewing or running with a friend. He can simply touch his friend's phone to share that context. Or, a user may wish to interact with a person and then decide on the information shared later. Upon tapping the phones together, either or both users can then launch an interaction based on a menu of possibilities then displayed, filtered to show applications that support peer-to-peer. The flexibility of either choosing the applications or the participants first allows the users to interact naturally depending on the context. Finally, we like to emphasize that it is not necessary for the receiving party to have pre-installed the code for the interaction. We can download the code on the fly, requiring no intervention other than

possibly an approval to download; this reduces the friction of interaction and helps the software go viral.

Self across interactive devices (e.g. phone to a PC). We now consider interactions running on multiple devices, each with their own input and output capabilities, but controlled by the same person. This kind of interaction is growing with the smart phones playing a more significant role in our life. We are spending more time and storing more information on the phone, but the phone is limited in its processing, input and output capabilities.

In this use case, it is the same person who decides on the interaction of interest. He may wish to have the PC assist with a task on the phone (e.g. to borrow the use of the PC keyboard for text entry), or to have the phone assist the PC (e.g. to ask the password manager on the phone to log the user into a web page using a challenge-response protocol). In either case, the user performs the same action by touching the PC with the phone, the contexts of the devices will be shared. The device with a sharable context is the initiator. Because the same person is controlling both ends, confirmations to accept invitations are unnecessary. The mode of operation is the same regardless of whether the user owns the PC since no setup is necessary.

We also envision in the future that we may wish to pair the phone with the PC for the entire duration the PC is used. Micro-interactions lead to long-lasting sessions, in which resources can be used across each device. Such usage patterns have been explored previously, for example using platform composition [14].

Remote control (e.g. phone to a TV or a car). This last use case refers to the control of other devices through a phone. Just like all the other cases, the micro-interaction may be initiated on either device. The user may wish to enlist another device to perform a task running on the phone. An example would be displaying a photo on the TV. Here, the phone is the initiator. Or, the device we wish to control may provide the context. For example, a driver may simply tap his phone on the car, the car will provide the context and request the driver's preference of music selection and seat setting. A device like a TV may have many possible modes of interactions, so it may wish to display a menu so that the user can pick the interaction of interest.

3 Protocols for Micro-Interactions

Simple interactions must be made as frictionless as possible, otherwise users will simply not be bothered. Take for example the simple task of running a collaborative whiteboard between two phones. First, the application must be available on both devices. If someone wants to run the whiteboard with another person, but the application is not yet installed, the friction involved in finding and downloading the application may overwhelm the benefits of running it. Second, the two users must join each other in a shared whiteboard session. Again, any investment beyond a few seconds in setting up the session may be too much for the users.

We would like to enable a "tap to share" experience for such collaborative applications. To minimize the setup overhead, we introduce protocols to address each of these aspects in collaboration:

– "first packet" delivery, to establish device communication
– code discovery and execution, and
– long-lasting multi-party runtime.

The micro-interaction enabled by combining these ideas is fast and intuitive: To run a whiteboard with a friend, the user simply launches the whiteboard program, while his friend turns on his device. They touch phones and the friend's phone tells him that he does not have the whiteboard application, prompting him to install it. He confirms, the application downloads and launches, joining the first user for an interactive session. If the friend already has the whiteboard application installed, the interactive session comes up in a couple of seconds, faster than it takes to even select the program from the list of available applications.

3.1 NDEF Exchange as "First-Packet" Delivery

Our user interactions are inspired by the NFC communication pattern as implemented in the Android OS's Gingerbread release [7]. Here, the P2P model restricts users to the exchange of an NDEF message in each direction. To ensure a fast interaction, application data is made available to the underlying operating system in advance of two devices interacting physically, and messages are reacted to with an asynchronous callback. An NDEF exchange can be contrasted to the HTTP or OBEX protocols, which provide a request/response interaction model. The lack of response during the NDEF exchange is especially important for NFC as we can program both active devices such as phones and passive devices such as stickers in a uniform way.

The NDEF data format is well-suited for this style of interaction. Each message is a compact representation of some well-defined payload. The type may be well known, such as a URI or MIME type, or of a type specially designed for some application.

3.2 Interaction Manifest for Cross-Platform Code Discovery

We define a simple data format called an *interaction manifest* that specifies the application to be run on the remote device. Our goal is to support platforms of any type (phone, PC, TV, etc.), thus the interaction manifest is defined as a MIME type consisting of *one or more* platform-specific application specifications. Each entry includes:

– A platform identifier, such as "Android", "iOS", or ''Web".
– A platform-specific application reference, uniquely identifying programs that are both installed on the phone or available online.
– An instance-specific application argument.

– An optional device modality, to support different application code for different device types.

The interpretation of an interaction manifest depends on the state of the remote device. If the remote device has a foreground application, it gets priority access to the interaction manifest. In the interaction manifest received comes from the same application, it acts according to the application argument provided. If it comes from a different application, it may still be able to understand the message if the applications are compatible. If there is no foreground application, or if the foreground application does not understand the message, the operating system handles the application manifest by launching the specified application. If the application is not already installed, the application for the appropriate platform will first be downloaded, with user confirmation.

For the whiteboard example, the application creates an application manifest that specifies where the source of the whiteboard application is located; the application manifest is presented to the remote device via NFC whenever two phones touch. Upon receiving the manifest, the remote operating system will launch the whiteboard application, after code download when necessary, if it is not already running.

3.3 Junction for Long-Lasting Application Sessions

Having the phones run the same application is the first step, how are they joined to the same session? For this, we use the Junction platform.

We have developed a platform called Junction as a way of maintaining a real-time application session across devices [8]. Junction applications can be contrasted to server-client programs. Under the server-client model, all devices connect to a central server, which manages the application's runtime and assets. Instead, Junction moves all application logic onto the end-devices.

Junction makes use of a communication hub called a switchboard. The switchboard does nothing but route messages to the devices in an application session. The session can be thought of as a chatroom in which all participants see all messages, and the server does nothing other than route messages to clients.

A session is represented uniquely with a URI. The URI encodes a unique session identifier as well as the switchboard's address. This URI acts as a capability for joining the application session, and is the application argument we use in the interaction manifest.

Junction is an abstraction and supports many implementations. For example, a session may be run through an XMPP chat server, locally over TCP/IP, or across phones using Bluetooth. Here, one phone acts as a hub, routing messages to other phones over Bluetooth sockets. Using Bluetooth, an application is run entirely locally, without any additional infrastructure.

4 CONTEXT SHARING

Tapping two programmable devices together creates a *symmetric* relationship, as each may try to provoke a response on the other device. This is very different from the familiar request-and-respond protocol where there is only one initiator. In this section, we discuss how two phones interact upon touching each other, with the assumption that both phones are NFC-enabled. We will relax this assumption in the next section as we show how to add NFC capabilities to legacy devices like PCs and TVs.

4.1 Context-Rich Interactions

We say that a device is *context-rich* if it is running a foreground application that wishes to share its context with a remote device. Otherwise, the device is *context-bare*. Because of the small display size, a smart phone has only one foreground application, which is the application whose interface is occupying the screen real estate. On a PC, the application with the cursor is the foreground application. The foreground application on the context-rich device registers an interaction manifest with the operating system, which is presented to the remote device whenever the phones touch.

The most straightforward combination is when a context-rich device touches a context-bare device, the former simply shares its context with the latter. In our earlier example, the phone that initiated the whiteboard application is the context-rich device. Touching it with a context-bare phone simply passes the whiteboard's interaction manifest to the latter. Consider, as a second example, a secure login application on the PC that uses the phone for challenge-response authentication. When the login application is in the foreground, the PC is context-rich, presenting to the phone an interaction manifest for authentication. The phone can then invoke its login application by simply tapping it to the PC.

If two context-rich devices come in contact, the respective interaction manifests are sent to the foreground applications, which may decide independently whether or not a received message is of interest, ignoring it otherwise. As an example, consider a device running a jukebox application (exposing its interaction manifest), and another browsing media files (exposing a file or link). The interaction results in the media file being sent to the jukebox, which opens the content, and a reference to the jukebox application made available to the browsing device, which ignores the message.

As a special case, consider two context-rich devices running the same application. We can easily establish a connection between the two devices for a long-lasting session, run over Bluetooth, Junction, or some other means. Each application indicates the connection information over which it can be reached. Because the exchange is symmetric, the devices must decide on which single session to use. They can come to an agreement by following a protocol based on the information they both have as a result of the exchange. For example, they can each generate a random number and agree to use the address given by the

device who generated the smaller value, as outlined in the Connection Handover framework [5].

4.2 Context-Bare Interactions

When two context-bare devices touch, each device defaults to sharing its *handover addresses* and device type (e.g. TV, PC, phone). The handover address provides an address the device can be contacted subsequently. It also uniquely identifies the device and can be used as a contextual cue. A device, upon receiving a handover address, may wish to present on its screen a menu of applications relevant to the device type of the second party. For example, we are likely to play a phone-to-phone game with another phone, but not with a TV. Most applications on our phone are not designed for multi-party use, and so even a basic filtering algorithm provides a meaningful context-aware application menu. We can also suggest recently used applications between us and the remote device, associating applications with either the device type or identity, which has been shown to be a useful contextual cue [21].

Whether or not to require user confirmation during an interaction depends on the device and situation. For example, direct input to applications running on a TV is cumbersome, and so our applications and multimedia run on a TV without confirmation. If security is of concern, the device can maintain a whitelist of devices that are allowed to open content. The "auto-answering" of our TV can make the difference between a compelling and unappealing end-user experience.

4.3 Labeling Arbitrary Objects with Contexts

We can also write out a context on an NFC tag and label any object or location with that tag. Consider the example we described earlier where we wish to set our phone to "night mode" when we go to bed. We can do so by launching the night mode application and writing the context that launches the application on an NFC tag by simply touching the phone to the tag. We then stick the tag on our nightstand, allowing us to simply place our phone over the tag to turn on night mode.

5 CONNECTION HANDOVER

As we discussed above, there are many compelling reasons for connecting our phone to other programmable devices like the PC, TV, and even the car. We wish to enable the "tap and run" experience of NFC without requiring the expense and effort of integrating an NFC radio into each device. We can easily add micro-interaction capabilities to existing networked devices with the help of a passive NFC tag, which can be purchased for about $1 a piece or for 20 cents in bulk. Furthermore, two devices without NFC radios can also share easily with the assistance of an active NFC device. Moreover, an application written to use NFC can support connection handover with non-NFC equipped devices without modification.

5.1 Handover Service

We add the NDEF exchange protocol to devices lacking NFC by running a simple listening service on the device. The service can be run on a PC, TV, or phone, with the exchange occurring over a Bluetooth or TCP/IP socket. The data exchanged between applications is the same as what would be exchanged over NFC.

Similar to HTTP redirects on the web, supporting connection handover requests does not require any changes in the application. The underlying platform detects the handover message, follows the handover protocol, and passes on the NDEF message to the application.

Bluetooth and TCP/IP can be interacted with at greater distances than NFC, so we must focus on the security and privacy of the handover exchange (although NFC too requires security considerations [6]). A first step is to deactivate the service when not in use. On a mobile phone, we turn off the service when the phone's screen is off, which also helps conserve battery. To prevent eavesdropping, we use public key cryptography to secure the message exchange. We can also use a whitelist to limit the devices that are allowed to interact with a service to prevent unwanted access.

5.2 Labeling the Devices

We associate a passive NFC tag with a supporting device. The tag can be affixed to the device, as we have done with a Nexus One phone in Fig. 2(a), or placed in a representative location, such as on a television remote control.

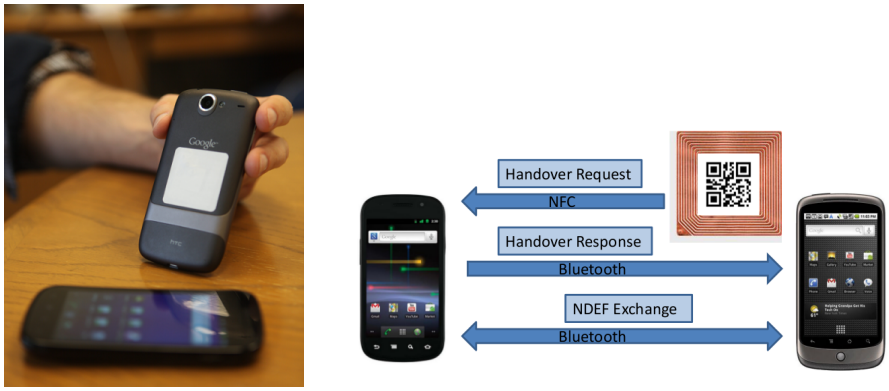

Fig. 2. The NDEF exchange handover protocol for devices lacking an NFC radio. (a), a connection handover sticker on a Nexus One. (b), the NDEF exchange handover protocol.

The tag stores the connection information for the NDEF exchange service, which must somehow be written. In our desktop and mobile applications, a

link allows the user to retrieve a QR code encoding the service details. Our application allows another device with an NFC radio and a camera to scan the QR code, convert the contents to NDEF, and write it to the tag.

5.3 Protocol

When an active NFC device scans a passive tag, it typically sends the contents to the foreground application. However, if the tag is an NDEF exchange handover request, the platform preempts the normal workflow, running the NDEF exchange handover protocol depicted in Fig. 2(b). The device establishes a connection over TCP/IP or Bluetooth, initiating the bidirectional NDEF exchange and handling the newly received NDEF message as if it had come directly from a tag or active device.

5.4 Information Transfer

We have seen how we can use a phone with NFC to interact with non-NFC devices. Using this technique, we can use our phone as a means of passing data between two devices lacking NFC, for example sending a multimedia file from a PC to a TV. We simply "pick up" the content from the first device with a touch, and "drop" it to a target device with a second touch. This is an intuitive way to transfer anonymous content [2]. The user does not need to start any application prior to "picking up" the content–in our system, the default handler of NDEF messages can be used to copy and paste content across devices.

For large files, we do not have to transfer the content to the phone when copying across devices. We simply transfer a pointer to the content and let the receiving device download it directly over Bluetooth, HTTP, or some other protocol.

We can also use the NDEF exchange protocol across devices without any NFC involvement and without modifying application code. We can write out the interaction manifest in a QR code, which can then scanned by a remote device. Since scanning a QR code can be cumbersome, our application also allows a user to recall previously used endpoints quickly.

6 IMPLEMENTATION

To explore the development and user experience of micro-interactions, we have developed an NFC abstraction layer for Android, an NDEF exchange handover service for Android and PC, and several applications.

6.1 NDEF Exchange Handover Request

Our NDEF exchange handover request was designed using the guidelines of the NFC Forum's Connection Handover specification [5], a general-purpose framework for setting up connections that run beyond the NFC radio. We use the static request mechanism of the profile.

6.2 EasyNFC library for Android

We have developed a library for NFC interactions on top of Android's core implementation, called EasyNFC. The library supports the NDEF exchange handover here described. To enable connection handover, the developer must request Internet or Bluetooth permission in their application, but the handover is invisible otherwise. EasyNFC also lets developers create a Bluetooth socket between devices easily. A developer simply implements the OnBluetoothConnected interface, with a callback triggered after establishing a Bluetooth connection.

6.3 LegacyNFC service for Android

The LegacyNFC application provides basic NDEF exchange functionality for Android phones that do not have an NFC radio. The application consists of a background service that runs whenever the screen is turned on. It listens over Bluetooth or TCP/IP for an NDEF exchange initiated by a peer, and also prepares local NDEF messages from applications, set using Android's system of Intents.

LegacyNFC can display the device's NDEF exchange endpoint information as a QR code. The QR code can be used to send messages immediately or stored by a remote phone for later use. The endpoint information can also be written to an NFC tag, supporting the fast NFC micro-interaction we discussed. An NFC-enabled phone can touch this sticker and initiate an NDEF exchange using the specified endpoint information.

When the user is running an application that uses the EasyNFC library, an icon appears in their notification bar. This UI element is created by LegacyNFC, requiring no extra work for the application developer. It allows the user to share their current application with other devices using NDEF exchange. The user can share the application via "QuickTap", following a stored NDEF exchange address, or by scanning a QR code. We envision other contextually-driven means of device selection.

6.4 DesktopNFC service for PCs and TVs

DesktopNFC is similar in nature to LegacyNFC, but is designed for use on PCs and TVs. The implementation is written in Java, and can support NDEF exchanges over both TCP/IP and Bluetooth. DesktopNFC has a console that allows basic functionality such as posting a URL or small file for use in an exchange. The daemon reacts to received NDEF messages automatically for content types deemed "safe", such as URLs and M3U playlist files. Otherwise, the user is prompted to manually handle the message. DesktopNFC can be run on a settop box attached to a TV. It interacts with a web browser running full screen, so content consumes the device when received.

7 APPLICATIONS

We have implemented a collection of different applications to explore the different use cases presented in this paper.

7.1 TapBoard

As discussed earlier, TapBoard allows a PC's keyboard to be used to enter text on a mobile phone (Fig. 1). Touching the phone to a PC opens a text box that is shared across devices. Using a PC's keyboard not only allows for faster typing, but also lets users copy and paste text from the PC to a phone with ease. This application relies on a connection handover since the PC does not have NFC natively and Junction for maintaining the long-lasting session. Since there is no remote software to install as the PC software is contained in a web page, we simply share a URL rather than an interaction manifest.

7.2 PocketSlides

A previously explored theme of micro-interactions is for a slideshow setup with minimal effort [13]. PocketSlides is our implementation that runs between a phone and a display. A user opens the presentation on her phone and sends this context to the display, as shown in Fig. 3. The display is written in HTML and listens for controls using Junction. When the display opens, the phone turns into a remote control to manage the display. The slideshow can be opened on any number of displays, with synchronized visualization. As with TapBoard, we share the display as a URL directly.

Fig. 3. A PocketSlides slideshow run between a phone and a display.

7.3 Hot Potato

Hot Potato is our mobile application for sharing multimedia over NDEF exchange. We hook into Android's "send" intent to allow the sharing of files and links from existing applications. For large files, we use a handover to HTTP or Bluetooth rather than relying on NFC. With Hot Potato, we can send files to other friends' mobile devices, open multimedia directly on a TV, or quickly send a file to a PC.

Hot Potato also has a "copy and paste" feature that supports picking up a file from one device and sending it to another. For large files, we can copy and paste a reference to the file, and transfer the data over a direct link. Fig. 4 shows how we can display a picture on our phone to a TV by touching an NFC-tagged remote control.

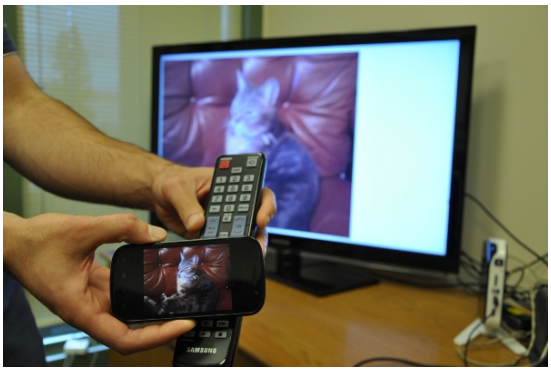

Fig. 4. Pushing multimedia to a remote display after touching a phone to a TV remote control.

7.4 weHold'Em

WeHold'Em is a game of poker played between phones and a TV. The game is built using Junction and installed on an Android phone. Touching phones together invites more players to the game, downloading the code if necessary. Touching the phone to a TV brings up the display, showing poker chips and community cards, as shown in Fig. 5. The TV's code is written in Javascript and HTML. Here, we make heavy use of the interaction manifest to specify where to download the mobile client and the software to run on the TV.

7.5 Musubi Exchange

Musubi is a social network run between phones. With a tap, users can exchange personas, including name, picture, contact information, as well as a public key for communication. The contacts enable the participants to engage in further

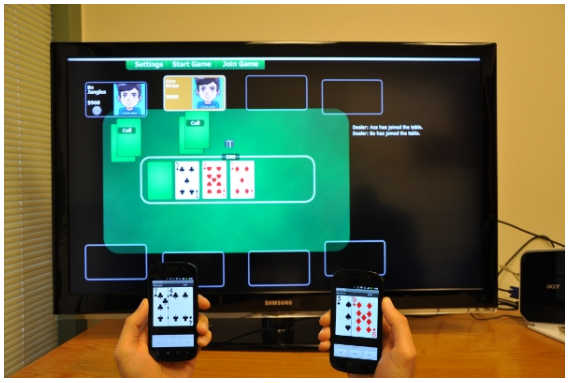

Fig. 5. A game of poker played between phones and a TV.

activities such as sharing photos or running applications such as a collaborative whiteboard or playlist. Musubi's exchange uses the interaction manifest with the hopes of making the application experience more viral.

Application	Description	Primitives
TapBoard	Use a desktop's keyboard to enter text on a mobile phone, entered in a simple text box application.	Phone-to-PC context transfer; Junction for long-lasting P2P sessions.
PocketSlides	A slideshow presentation controlled by a mobile phone and displayed on a TV or projector.	Slideshow persisted on phone; Junction for long-lasting sessions.
Hot Potato	Send files, links, and text from a phone to another phone, pc, or TV using NDEF exchange.	Android hooks via "send" intent; auto-answer on TV devices; handover for large files.
weHold'Em	A game of poker played between mobile phones and a TV. Phones are used as player controllers and the TV is a communal display.	Interaction manifest for cross-device and cross-platform support; Junction for long-lasting session.
Musubi Exchange	Touch phones to exchange profile information for a social network, including name, contact information, and public key.	Interaction manifest for viral bootstrapping.

Fig. 6. Applications featuring micro-interactions.

7.6 Summary

While many of these applications described above are not new, they were all built using the framework described in this paper. As such, we demonstrate that the framework is general enough to support a wide variety of applications.

Each primitive is reused in several applications. Furthermore, our development experience suggests that it is relatively easy to develop such applications.

These applications do not require any pre-configuration, unlike other technologies like Bluetooth pairing. There are no buttons to click or words to type in, except occasionally the user has to approve a software download.

8 RELATED WORK

The proliferation of our digital devices has drawn much attention to how we can use these devices in concert [12]. Many technologies have been created to support cross-device interactions, offering different solutions at each level of the stack [18]. Research has also shown that users frequently employ cross-device habits, and that the amount of configuration involved can drastically affect the usability of the system [3]. Data privacy is also a source of concern for many, which our applications embody using NFC and Junction in their runtimes.

Devices can be composed in a number of ways. For example, platform composition allows devices to share resources at the platform level [14], activity-based computing revolves applications around high-level tasks [1, 4], and devices can be organized around a single owner [15]. Our focus on micro-interactions creates ad-hoc compositions in an instant, with context inferred from the interactions themselves.

The difficulties associated with pairing devices has been the focus of much research [9, 17, 20, 23]. Micro-interactions require a pairing and service discovery process that is essentially instantaneous. The popular Bump service for the Android and iPhone platforms connects two users by mapping accelerometer readings and other environmental data on a cloud-based service [22]. All data is exchanged through this service. A major benefit of our NFC and NDEF exchange handover techniques is the interactions run purely locally, avoiding privacy concerns and also providing a faster, more robust user experience. Bump also must be run in the foreground, taking over the phone's UI, while NFC runs in the background, "behind the screen."

Using RFID and QR codes as a way to bridge the physical and digital worlds has been explored at length, and in particular, using a phone as the point of interaction [11, 13, 24]. In particular, the Elope system uses an RFID tag to set up a connection and also initiate an action associated with that tag. Instead, we use the tag as a contextual cue for determining which application to invoke, and combine it with the context derived from the phone itself.

9 Conclusions

As smart phones are fast becoming a part of our digital self, we believe that micro-interactions on our smart phones will have a significantly impact on our daily life. This paper shows how we can provide a consistent "tap-and-share" interface beyond what is natively supported by NFC.

This paper presents three useful ideas for developing micro-interactions. First, we propose interaction manifest as a universal descriptor of a multi-party application that can be run across multiple platforms. This concise descriptor can be embedded in an NDEF message, which can be transmitted either with NFC or other medium to enable remote participation. The Junction application platform facilitates the development of decentralized multi-party applications, by providing support for invitations, download of software, as well as a messaging service through local or remote switchboards. Finally, the concept of connection handover for NDEF exchange allows NFC-enabled applications to run on devices lacking NFC, unmodified. Our experience shows that these primitives make writing compelling micro-interactions easy and the resulting applications are simple and intuitive to use.

It is exciting that NFC is now available on the latest smart phones for which a healthy ecosystem of third-party applications already exists. We believe that the abstractions contributed by this paper will help promote the development of useful micro-interactions in the market place, and we have made all of the software discussed available as ppen source to help reach that goal. With the prediction that one in five smartphones worldwide will be NFC capable by 2014 [16], the vision of having many consumers using micro-interactions regularly, without even thinking about it, may soon become a reality.

10 Acknowledgments

We would like to thank Aemon Cannon, Chanh Nguyen, T. J. Purtell, and Ian Vo for their help in developing the applications described in this paper. This research is supported in part by the NSF POMI (Programmable Open Mobile Internet) 2020 Expedition Grant 0832820, NSF grant CCF-0964173, Stanford Clean Slate Program, and Stanford MobiSocial Computing Laboratory.

References

1. E. Bardram. Activity-based computing: support for mobility and collaboration in ubiquitous computing. *Personal Ubiquitous Comput.*, 9:312–322, September 2005.
2. Richard A. Bolt. "put-that-there:" voice and gesture at the graphics interface. In *Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '80, pages 262–270, New York, NY, USA, 1980. ACM.
3. David Dearman and Jeffery S. Pierce. It's on my other computer!: computing with multiple devices. In *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, CHI '08, pages 767–776, New York, NY, USA, 2008. ACM.
4. W. Keith Edwards, Mark W. Newman, Jana Z. Sedivy, and Trevor F. Smith. Experiences with recombinant computing: Exploring ad hoc interoperability in evolving digital networks. *ACM Trans. Comput.-Hum. Interact.*, 16(1):1–44, 2009.
5. NFC Forum. Nfc forum technical specifications, 2010. `www.nfc-forum.org/specs/spec_list`.

6. Lishoy Francis, Gerhard Hancke, Keith Mayes, and Konstantinos Markantonakis. Practical nfc peer-to-peer relay attack using mobile phones. In *Proceedings of the 6th international conference on Radio frequency identification: security and privacy issues*, RFIDSec'10, pages 35–49, Berlin, Heidelberg, 2010. Springer-Verlag.
7. Google. Near field communication, 2011. `developer.android.com/guide/topics/nfc/index.html#p2p`.
8. Junction. `openjunction.org`.
9. Rene Mayrhofer and Hans Gellersen. Shake well before use: Intuitive and secure pairing of mobile devices. *IEEE Transactions on Mobile Computing*, 8:792–806, 2009.
10. Dan R. Olsen, S. Travis Nielsen, and David Parslow. Join and capture: A model for nomadic interaction. In *In Proceedings of 14th Annual ACM Symposium on User Interface Software and Technology*, pages 131–140. Press, 2001.
11. Eamonn O'Neill, Peter Thompson, Stavros Garzonis, and Andrew Warr. Reach out and touch: using nfc and 2d barcodes for service discovery and interaction with mobile devices. In *Proceedings of the 5th international conference on Pervasive computing*, PERVASIVE'07, pages 19–36, Berlin, Heidelberg, 2007. Springer-Verlag.
12. Antti Oulasvirta and Lauri Sumari. Mobile kits and laptop trays: managing multiple devices in mobile information work. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '07, pages 1127–1136, New York, NY, USA, 2007. ACM.
13. Trevor Pering, Rafael Ballagas, and Roy Want. Spontaneous marriages of mobile devices and interactive spaces. *Commun. ACM*, 48(9):53–59, 2005.
14. Trevor Pering, Roy Want, Barbara Rosario, Shivani Sud, and Kent Lyons. Enabling pervasive collaboration with platform composition. In *Proceedings of the 7th International Conference on Pervasive Computing*, Pervasive '09, pages 184–201, Berlin, Heidelberg, 2009. Springer-Verlag.
15. Jeffrey S. Pierce and Jeffrey Nichols. An infrastructure for extending applications' user experiences across multiple personal devices. In *Proceedings of the 21st annual ACM symposium on User interface software and technology*, UIST '08, pages 101–110, New York, NY, USA, 2008. ACM.
16. Juniper Research. 1 in 5 Smartphones will have NFC by 2014, 2011. `http://www.msnbc.msn.com/id/42584660/ns/business-press_releases/`.
17. Nitesh Saxena, Md. Borhan Uddin, and Jonathan Voris. Universal device pairing using an auxiliary device. In *Proceedings of the 4th symposium on Usable privacy and security*, SOUPS '08, pages 56–67, New York, NY, USA, 2008. ACM.
18. Bill N. Schilit and Uttam Sengupta. Device ensembles. *Computer*, 37:56–64, December 2004.
19. Dominik Schmidt, Fadi Chehimi, Enrico Rukzio, and Hans Gellersen. Phonetouch: a technique for direct phone interaction on surfaces. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, UIST '10, pages 13–16, New York, NY, USA, 2010. ACM.
20. Frank Stajano and Ross J. Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. In *Proceedings of the 7th International Workshop on Security Protocols*, pages 172–194, London, UK, 2000. Springer-Verlag.
21. John C. Tang, James Lin, Jeffrey Pierce, Steve Whittaker, and Clemens Drews. Recent shortcuts: using recent interactions to support shared activities. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '07, pages 1263–1272, New York, NY, USA, 2007. ACM.

22. Bump Technologies. `bu.mp`.
23. Ersin Uzun, Kristiina Karvonen, and N. Asokan. Usability analysis of secure pairing methods. Technical report, In Usable Security (USEC, 2007.
24. Roy Want, Kenneth P. Fishkin, Anuj Gujar, and Beverly L. Harrison. Bridging physical and virtual worlds with electronic tags, 1999.
25. Mark Weiser. The computer for the 21st century. *Scientific American*, 265(3):66–75, September 1991.
26. Andrew D. Wilson and Raman Sarin. Bluetable: connecting wireless mobile devices on interactive surfaces using vision-based handshaking. In *Proceedings of Graphics Interface 2007*, GI '07, pages 119–125, New York, NY, USA, 2007. ACM.