P2P 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 introduces micro-interactions. Micro-interactions are categorized by their nearly instantaneous occurrence. They are invoked quickly and naturally, requiring no pre-configuration or setup. They occur with an absolute minimum amount of friction and require no training.

We focus on device-spanning micro-interactions based on NFC (Near-Field Communication), exploring their properties and impact. The reduction in friction enables a host of applications that would otherwise be impractical for real-world use.

This paper discusses the micro-interaction paradigm, introduces several useful programming abstractions for it, and presents a number of applications that we have built using those abstractions. We demonstrate that by focusing on micro-interactions, our mobile phones can provide a single focal point of our digital assets so we can easily share 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.

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 peer-to-peer micro-interactions; they will not be used unless they are as frictionless as micropayments.

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. Unlike dedicated NFC cards, not only can phones scan in information, but also programatically generate new information for other devices. Furthermore, information received can be processed automatically by one of the many available applications on the phone, using the NFC type definitions combined with operating system-level dispatch. 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 facilitate peer interactions. They can communicate directly without requiring a third-party server. The effortless connection of NFC opens up many opportunities for the phones to be used to enhance physical social encounters.

Micro-Interactions on the Phone
The concept of device-to-device micro-interactions has been explored by many researchers [24, 18, 13, 9]. 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?
As an example, 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.

Based on the ideas we present in this paper, we have developed an application called TapBoard that supports such a micro-interaction. With TapBoard, the user simply touches the phone with an NFC tag on the keyboard of a PC, the PC will bring up a webpage with a simple text box. 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.

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 difficulties 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.

Contributions
This paper makes the following contributions:

1. We present a large number of scenarios where micro-interactions 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. 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 (contact exchange, data sharing, keyboard sharing, remote presentation, and a multi-party poker game) using the libraries. Our experience suggests that the abstractions are powerful in simplifying the development of micro-interactions.

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.

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.

At Home The phone is commonly used as an alarm clock. Putting the phone on our bedside table puts the phone in “night mode,” silencing our non-critical notifications such as when we receive emails.

A digital photo frame can now be set at a touch. We look at a photo on our phone, and touch it to the frame to set it. We can also look at a contact on our phone to grant them remote access to set photos for us.

In the living room, we touch our phone to our TV remote to turn our phone into a digital remote. We can change channels by browsing or searching or even use 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 sends 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.

Figure 1: Using TapBoard to enter mobile text.
When we are visiting another company, we need a guest badge. On our phone, we open the invitation from our host and touch it to the kiosk at the receptionist’s desk to share who we are and who we are meeting. As we wait to meet our host, an email comes in. 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 other phones in the room, sending a copy of our slides.

**Out and About** On our way out, we park with the valet. 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 head to lunch, and “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 you’re on your way back, so the car is ready when we return.

Later, we take the bus to a ballgame. We touch our phone to a sign at the bus stop to get the schedule of when the next bus is coming. At the game, we touch our phone to the attendee’s kiosk to submit our ticket. In response, our phone lets us launch an app for the stadium. We can connect to other fans in the ballpark and watch instant replays. It knows our seat number and lets us order concessions directly.

Even later, we head to a 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, in case we forget to ask for it later. There’s a TV running at the party, 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.

**Kinds of Micro-interactions**

We can organize all the various micro-interactions described based on 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 peer. 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 having a shareable 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. The files on the phone may be made available on the PC, no different from how network-mounted file systems work. For example, the browser may use the bookmarks saved on the phone. Similarly, we can unify the screens and I/O devices so we can use the keyboard on the PC for SMS messages by moving the pointer onto the phone. Instead of requiring individual applications providing cross-device support, such capability is best provided at the operating system level so it can be made available to all applications. Such usage patterns have been explored previously, for example using platform composition [13]. Micro-interactions allow for an easy and intuitive way to set up the connection.

**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.
P2P MULTI-PARTY APPLICATIONS
We now consider the first use case where two phones wish to share a multi-party application. Consider the task of running a collaborative whiteboard across two mobile phones. Because of the simple nature of the program, it is unlikely to be run if it takes even a moderate amount of effort to initiate. However, if the instantiation of the program relies on micro-interactions, it can prove compelling.

There are a few critical challenges in running the whiteboard program. 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 introduce a few key components that, when combined, minimize the overall effort required to initiate the whiteboard and other multi-party applications. We focus on three key concepts:

- “First packet” delivery, to establish device communication
- Code discovery and execution
- Long-lasting multi-party runtime

The micro-interaction enabled by combining these ideas is fast and intuitive: A user wants to run a whiteboard with a friend. He launches the program, while the friend turns on his device. They touch phones, and the friend is told he does not have the whiteboard application, but is prompted to install it. He confirms, and the application downloads and launches, joining the first user for an interactive session.

If the friend already has the whiteboard application installed, it takes under five seconds for him to join the application session, less time than it takes to even select the program from the list of all available applications.

NDEF Exchange over NFC for “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 bidirectional exchanging of a single NDEF message. 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 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.

Modern mobile platforms are characterized by having a single foregrounded application at a time. During an NDEF exchange, the foregrounded application gets priority access to sending and receiving messages. Otherwise, a received message can be handled by another application according to the platform’s dispatch system.

Interaction Manifest for Cross-Platform Code Discovery
During an NDEF exchange, our goal is to provide data that can be understood by any visiting device. That device may be of any type (phone, PC, TV, etc.), and may have a foreground application running or not.

We define a simple data format called an Interaction Manifest to represent an application in a way that can be understood regardless of the visiting device while avoiding a request/response interaction.

The Interaction Manifest is 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, identifying both installed and installable programs.  
- An instance-specific application argument. 
- An optional device modality, to support different application code for different device types.

When a remote device receives an Interaction Manifest, it can be understood in any context—A foregrounded application can determine if the message is from the same application on a remote device and handle the application argument; a platform can locate an appropriate application to launch, regardless of whether or not it has been installed. A foregrounded application of an alternate type may also understand the message if the applications are developed cooperatively.

The previously mentioned whiteboard application is available for Android and the iPhone, and each platform has its own way of referring to locally installed applications and for locating these applications online, which we encode in the Interaction Manifest. Additionally, the whiteboard can be displayed on a TV using its web implementation, also encoded in the manifest.

Junction for Long-Lasting Application Sessions
Returning to our whiteboard example, we have now established how the phones can identify each other and initiate the application, downloading the code if necessary. The Interaction Manifest allows us to pass an arbitrary application argument between instances. We use this argument to establish a session across the devices.

We have developed a platform called Junction as a way of maintaining a real-time application session across devices. 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.

**CONTEXT SHARING**

Tapping two programmable devices together creates a “symmetric” relationship, as each may transfer context information to the other party, hoping to provoke a response. This is very different from the familiar request-and-respond protocol where there is only one initiator. This section discusses how we can determine the context to be shared across interacting devices.

We assume in this section that all devices 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.

**Context-Rich Interactions**

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. If this foreground application has registered with the operating system that it wishes to use the NDEF exchange interface, we say the the phone is context-rich. Otherwise, the device is context-bare.

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 message is sent to the foreground application, which may decide independently whether or not a received message is of interest, discarding 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 discards 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].

**Context-Bare Interactions**

When two context-bare devices touch, it defaults to having the devices share handover addresses and device types (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 [20].

Once an application is launched from a menu, the application can provide the context of subsequent interactions, thus streamlining the experience. Suppose you are at a party with many new people you wish to make contacts with. The first time you tap your phone with another, you have to explicitly launch the contact-sharing application from a menu. The application will provide the context for subsequent greetings with new friends.

Whether or not to require user confirmation before completing an interaction depends on the device and situation. For example, direct input to applications running on a TV are cumbersome, and so our applications and multimedia run 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 non-compelling end-user experience.

**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, write the context to launch the application on an NFC tag by simply touching the phone to the tag, then stick the tag on our nightstand. We can simply
place our phone over the tag to easily turn on night mode. The night mode can be turned off, for example, as we turn off the alarm.

**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 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.

**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.

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. Additionally, we can use a whitelist of devices that are allowed to interact with a service to prevent unwanted access.

**Labeling the Devices**

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

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.

**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 Figure 3. 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.

**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.

Our application also supports an alternative workflow. If the remote device features an NDEF exchange endpoint, A user can touch the device first and then select content afterwards. This is useful in sending multiple pieces of content to the remote device; the device is touched once, and then each message is scanned and sent in serial.

We can also use the NDEF exchange protocol across devices without any NFC involvement. We simply use the QR code directly for addressing the remote device. Scanning a QR code can be cumbersome, but our application allows a user to recall previously used endpoints quickly. Figure 5 shows how the whiteboard application can be shared between de-
VICES without NFC.

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.

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.

We also wanted to support menu selection when a handover tag is detected. However, a handover request message cannot be automatically handled by Android’s dispatch system—we require either a MIME type or URI. We created an NDEF protocol to convert an NDEF message into a URI. We simply use the “ndef://” scheme and use a Base64 encoding of the NDEF message as our path. The authority encodes the type of the message, here “wkt:hr” for the well-known-type Handover Request. Using a URI also allows the NDEF to be embedded in a QR code, for which a MIME type would be less suited.

EasyNFC library for Android
We have developed a library for NFC interactions on top of Android’s core implementation. Our primary goals are (1), to provide compatibility across mobile platforms where possible, and (2), to support non-NFC equipped devices using the NDEF exchange connection handover. Secondarily, we aim to simplify NFC development and provide a useful set of common primitives for developers.

The connection handover occurs with little worry by the developer. The developer must request Internet or Bluetooth permission in their application, but the handover is invisible otherwise (although it can be disabled).

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.

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, listening over Bluetooth for an initiated NDEF exchange. The daemon interacts with applications using intents for IPC.

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 code can also be written to an NFC tag, supporting the fast NFC micro-interaction we’ve discussed.

If LegacyNFC receives an NDEF exchange request, the daemon first checks to see if the foreground application is interested in an NDEF exchange. If so, any received NDEF is passed to the application using a broadcasted intent, and an outbound message is sent if available. Otherwise, LegacyNFC decodes a received NDEF message and presents it to the user as a notification, allowing him to open a URL or application, as seen in Figure 4.

When the user is running an application that uses the EasyNFC library, an icon appears in their notification bar as shown in Figure 5. This UI element is created by LegacyNFC, with no extra work for the application developer. It allows the user to share their current application with other devices over 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.

Figure 4: Receiving a message via NDEF Exchange using LegacyNFC.

Figure 5: Using LegacyNFC to share a whiteboard application.

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”,

Figure 4: Receiving a message via NDEF Exchange using LegacyNFC.

Figure 5: Using LegacyNFC to share a whiteboard application.
such as URLs and M3U playlist files. Otherwise, the user is prompted to manually handle the message.

We run DesktopNFC on a settop box attached to a TV. It interacts with a web browser running full screen, so content consumes the device when received.

**Activity Director**

Additionally, we have created an application called the Activity Director to handle application invocations from an interaction manifest. On Android, we handle the interaction manifest MIME type, received over NFC or any other means. The Director checks to see if the application is installed locally, and launches it if so, passing in the application argument. Otherwise, the application can be downloaded from the Android Market or from the web. On the desktop, we’ve bundled the Director with DesktopNFC, handling interaction manifest messages received from our NDEF exchange.

**APPLICATIONS**

We have implemented a collection of different applications to explore the different use cases presented in this paper. TapBoard and PocketSlides allow one to use a PC in conjunction with the user’s phone. Hot Potato supports transfer of information between multiple parties as well as to the TV. weHold’Em and DungBeetle are both multi-party interactions; both applications leverage the TV in addition to the individuals’ phones for communal display of information.

**TapBoard**

As discussed earlier, TapBoard allows a PC’s keyboard to be used to enter text on a mobile phone (Figure 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.

**PocketSlides**

A previously explored theme of micro-interactions is for a slideshow setup with minimal effort [12]. PocketSlides is our implementation that runs between a phone and a display. A user opens the presentation on her phone and touches the phone to a tag representing the display. 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.

**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 our NDEF exchange link. 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.

Figure 7 shows how we can display a picture on our phone to the display by simply touching the remote control, at the back of which is an NFC tag with the TV’s NDEF exchange address. Here, we’ve additionally modified Android’s bundled Gallery3D application, automatically sharing the on-screen image with a device over NDEF exchange.

**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. The TV’s code is written in Javascript and HTML. Here, we make heavy use of the interaction manifest. We use it to specify where to download the mobile client and to specify a different piece of software to run on a TV.

**DungBeetle Exchange**

DungBeetle is a social network run between phones. With a tap (Figure 9), users can exchange personas, including name, picture, contact information, as well as a public key for com-
munication. The contacts enable the participants to engage in further activities such as sharing photos (Figure 10) or collaboratively creating a playlist (Figure 11). DungBeetle’s exchange uses the interaction manifest with the hopes of making the application experience more viral.

Figure 8: A game of poker played between phones and a TV.

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. In a sense, the invitation to download software itself is a personal recommendation that is often welcome, considering the difficulty of finding useful software among the large titles available.

Since micro-interactions have almost no user interaction to speak of, we do not feel that we need to conduct a formal user study. User studies of the applications themselves are outside the scope of this paper. We have shared the applications with friends and students in a class informally. They are generally impressed by the ease of use. Videos of our demos submitted to YouTube have attracted over 20,000 views, suggesting perhaps there is consumer interest in this technology.

RELATED WORK
The proliferation of our digital devices has drawn much attention to how we can use these devices in concert [11]. Many technologies have been created to support cross-device interactions, offering different solutions at each level of the stack [17]. 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 [13], activity-based computing revolves applications around high-level tasks [1, 4], and devices can be organized around a single owner [14]. Our focus on micro-interactions allows ad-hoc compositions to be created in an
Primitives
- Use a desktop’s keyboard to enter text on a mobile phone, entered in a simple text box application.
- A slideshow presentation controlled by a mobile phone and displayed on a TV or projector.
- Send files, links, and text from a phone to another phone, PC, or TV using NDEF exchange.
- A game of poker played between mobile phones and a TV.
- Touch phones to exchange profile information for a social network service through local or remote switching support for invitations, download of software, as well as development of decentralized multi-party applications, by providing a consistent “tap-and-share” interface to be used as long as one of the interacting parties has an active NFC radio. And, with no modifications to the applications, we can still support sharing using QR codes across devices with no NFC capabilities whatsoever.

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 smartphones 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 released all of the software discussed as Open Source to help reach that goal. With the prediction that 1 in 5 smartphones worldwide will be NFC capable by 2014 [15], the vision of having many consumers using micro-interactions regularly, without even thinking about it, may soon become a reality.

### References


