A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of Master of Science in
Computer Science and Engineering

by
Vamsi Krishna Raju, Alluri
Dr. Sergiu Dascalu / Thesis Advisor
December 2015
We recommend that the thesis prepared under our supervision by

VAMSI KRISHNA RAJU, ALLURI

Entitled

User-Controlled Information Transfer Between Digital Devices With A Gestural Input Component

be accepted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Sergiu Dascalu, Ph.D., Advisor

Frederick C Harris, Jr., Ph.D., Committee Member

Veronica Dahir, Ph.D., Graduate School Representative

David W. Zeh, Ph.D., Dean, Graduate School

December, 2015
ABSTRACT

There is an interplay between the messages users share electronically with others and the transformations that take place in the relationships between them. Every message that is shared has the potential to strengthen or break these relationships. Digital applications should reflect these changes in relationships effectively whenever they occur and preferably with the user who wants to see a change being in direct control. Today, in widely used tools like email, users could very easily lose control over who sends messages to their inbox and the situation quickly turns into a game of whack-a-mole that then lasts forever. Our goal is to explore this problem space without relying on existing digital infrastructure to find a solution that allows users to be in direct control of their relationships and improve their user experience. In essence, we improve the delete button in a contacts application from one that just deletes a string of stored data into another that can terminate relationships effectively.

In this thesis we first propose the “i80” system which demonstrates how this control is possible using distinct tokens to represent relationships between users. The server backend of i80 takes care of managing a user’s relationships with others and also ensures messages are delivered in real-time between valid relationships. i80 also has two front-end applications to ensure users can receive messages on personal computers and smartphones. We leverage QR codes to simplify the transfer of tokens and quickly create relationships between two or more users.

Also, gestures like swipe-down-to-refresh on smartphones have improved our user experience dramatically and we have applied a similar pattern to improve the experience of sharing files with others. Thus, we also propose the Pinch application to demonstrate how users can use a single pinch gesture to share files with others instead of selecting multiple buttons from a menu on a smartphone.

We have built prototypes of i80 which contains a server backend to manage rela-
tionships and transfer files across multiple users and devices, front end web and mobile applications to demonstrate common actions like creating relationships and sharing files, and a Pinch-enabled file manager to share files with others. We have evaluated both i80 and Pinch to demonstrate that the user experience of sending files is improved while simultaneously giving users better control over both their relationships and data.
To my parents
ACKNOWLEDGEMENTS

Firstly, I would like to express my sincere gratitude to my thesis advisor Dr. Sergiu Dascalu for his patience, motivation, and continuous support and advice throughout my Master's and especially for this thesis.

Besides my advisor, I would like to thank Dr. Frederick C. Harris and Dr. Veronica Dahir for serving on my thesis committee and for providing feedback on this work.

I would also like to thank Dr. Veronica Dahir again for supporting me with a graduate assistantship at the Center for Research Design and Analysis for some time during my Master’s which has been a huge help. I would also like to thank Mary Anne Christensen at the Disability Resource Center in UNR both for giving me the opportunity to work there and for the many things that I have learned only at this place.

I would like to thank my wonderful team members in the Analytics Product Management team at SAP who have both made my work fun over the past year and left me with enough energy to finish this thesis along with a full-time job.

Sincere thanks to my friends Avni and Vijeth, whose thoughtful advice and support have helped me navigate a new chapter in my life effortlessly.

A huge shout out to all my friends at UNR, I will cherish all our memories from this place and our friendships long into the future.

Finally, I would like to thank my parents and sister for their boundless love and support through all these years.
# CONTENTS

Abstract ................................................................. i  
Dedication ............................................................... iii 
Acknowledgements ....................................................... iv 
Contents ................................................................. v  
List of Tables ........................................................... vii 
List of Figures .......................................................... viii  

1 Introduction 1  
1.1 Motivation .......................................................... 3  
  1.1.1 Control ......................................................... 3  
  1.1.2 Gestures ....................................................... 5  
  1.1.3 Information .................................................. 6  
1.2 Challenges and Issues ............................................. 7  
1.3 Proposed Solution ................................................ 9  

2 Background and Related Work 11  
2.1 Gesture .............................................................. 11  
2.2 Pairing Users ...................................................... 16  
2.3 Credit Card Tokenization ......................................... 18  
2.4 Control over Relationships in Federated Networks .......... 19  
  2.4.1 Email ......................................................... 19  
  2.4.2 XMPP ........................................................ 20  

3 Proposed Solution 22  
3.1 Pinch Gesture ....................................................... 22  
3.2 User Relationships ................................................ 24  
3.3 Information Payload .............................................. 29  
3.4 Privacy .............................................................. 30  
3.5 Usage Scenarios ................................................... 31  
3.6 Federated Design .................................................. 32  
3.7 Next chapters ...................................................... 35  

4 Specification and Design 36  
4.1 Introduction ....................................................... 36  
4.2 Goals ................................................................. 36  
4.3 Assumptions ....................................................... 37  
4.4 Features and Functionality ..................................... 37  
4.5 Architecture ....................................................... 39  
  4.5.1 Backend ...................................................... 39  
  4.5.2 Clients ....................................................... 42  
  4.5.3 Payload Delivery .......................................... 43
## Prototype Implementation

5.1 i80

5.1.1 i80 Backend

5.1.2 i80 Mobile Frontend

5.1.3 i80 Web Frontend

5.2 Pinch

## Professional Feedback and Evaluation

6.1 Experimental Evaluation

6.1.1 Test Plan and Materials

6.1.2 Results

6.2 Comparative Evaluation

## Future Work and Conclusions

Bibliography
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Available API calls for Link Management on i80</td>
<td>51</td>
</tr>
<tr>
<td>5.2</td>
<td>Available API calls for User Management on i80</td>
<td>52</td>
</tr>
<tr>
<td>5.3</td>
<td>Available API calls for Device Management on i80</td>
<td>52</td>
</tr>
<tr>
<td>5.4</td>
<td>Available API calls for Payload Management on i80</td>
<td>53</td>
</tr>
<tr>
<td>6.1</td>
<td>Tasks performed by participants on i80 and Pinch</td>
<td>68</td>
</tr>
<tr>
<td>6.2</td>
<td>After scenario questionnaire results</td>
<td>71</td>
</tr>
<tr>
<td>6.3</td>
<td>Perceived usefulness and ease of use questionnaire results</td>
<td>72</td>
</tr>
<tr>
<td>6.4</td>
<td>Comparative evaluation of features of i80, Messaging services, File sharing services and Email</td>
<td>75</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

1.1 Email id obfuscation by authors of a paper on spam [28] .................. 2  
1.2 The new contact management application on Android smartphones [10] 4  
2.1 Gmail web application [12] ....................................................... 12  
2.2 Gmail smartphone application [11] ............................................. 12  
2.3 Swipe down to refresh gesture in a smartphone application [29] ...... 13  
2.4 Swipe right to archive an item gesture in a smartphone application [18] 14  
2.5 Shake to send feedback gesture in a smartphone application [13] ... 15  
2.6 A sample QR code ................................................................. 17  
3.1 A graph to outline the exponential increase in the number of steps when sharing with multiple applications. .......................... 23  
3.2 Approach architecture and design ............................................... 27  
3.3 Two users exchanging details to establish a new link between them . 28  
3.4 Using the link to transfer a file .................................................. 29  
3.5 One user removing the link to stop receiving files ....................... 29  
3.6 Users communicating with others in a federated network. ............ 33  
4.1 Sequence diagram 1. ............................................................... 37  
4.2 Sequence diagram 2. ............................................................... 38  
4.3 Software architecture with components of i80 and Pinch .................. 40  
5.1 Various components in the i80 backend ...................................... 46  
5.2 i80 and Pinch Mobile applications ............................................ 55  
5.3 i80 user login screen ............................................................. 56  
5.4 Create and Join Link screen .................................................... 57  
5.5 New payload received notification .............................................. 58  
5.6 Login page .......................................................................... 58  
5.7 Main dashboard page ............................................................ 59  
5.8 Dialog box with form for Create Link ........................................ 59  
5.9 Dialog box with QR code scanner for Join Link .......................... 60  
5.10 Set new active link from list of links ....................................... 61  
5.11 Verify active link in permanent notification ................................. 62  
5.12 Browse folders in the Pinch application .................................... 63  
5.13 Select file and do pinch gesture ............................................... 63  
5.14 File uploaded toast notification .............................................. 64  
6.1 After scenario questionnaire .................................................... 67  
6.2 Perceived usefulness part of the PUEU questionnaire ..................... 69  
6.3 Perceived ease of use part of the PUEU questionnaire .................... 70  
6.4 List negative and positive aspects part of the PUEU questionnaire ... 70  
6.5 Step diagram showing the differences in the number of user actions required to share information. ....................... 77
CHAPTER 1

INTRODUCTION

Even today, after years of development, our experience with digital devices and sharing information is not completely pleasant. We are not able to filter spam one hundred percent from our incoming email [15] and we have mostly accepted the fact that we may inevitably lose an important message to the spam filter every once in a while. We look for innovative ways to obfuscate our email ids to ensure they do not get into the hands of spammers when we post them publicly. We also take pains to manage multiple ids to communicate with users with different levels of trustworthiness.

Also, sometimes in search for better control over our digital relationships, we routinely place our data in the hands of entities that have control over the whole network [21]. We inevitably lose control over our own data and it is either tough or impossible to claw control back from these entities. A federated network like email on the other hand can provide us with control over data we end up losing control over relationships once we share our email ids with others. Having absolute control over both digital relationships and data is very important and we should not be forced to resort to workarounds or degrade our existing user experience to achieve them.

To find a solution to solve both these problems, this thesis introduces “i80” to allow users to be in control of their short term relationships without giving others permanent credentials like email ids and phone numbers that are valid for a longer term.

It also introduces a novel take on the common pinch gesture we use with our smartphones and repurposes it to enable users to share information without selecting options from a menu. It briefly highlights how a tool that allows data transfer and is
Figure 1.1: Email id obfuscation by authors of a paper on spam [28]

agnostic to the type of data being transferred could be beneficial to users as it would allow multiple clients to co-exist on the same infrastructure catering to different needs of users.

The first interstate freeway was a proposed take on transportation compared to the then prevalent railroads. We named our system “i80” because, like the interstate freeway, we envision a different abstraction layer to digitally connect users and transport information payloads between their computing devices. Users or groups of users are connected with links that a wide variety of digital applications can use as a common layer to transport their information payloads through.
1.1 Motivation

1.1.1 Control

Whenever there is wide adoption of a new technology by a large percentage of users, it opens up scope to rethink many standard practices that were designed for a set of devices with different capabilities. If we can redesign these old practices properly to take advantage of the new reality, where devices by default have more features and capabilities, users can get a better experience.

Users use an id to identify themselves uniquely when they use digital tools. For example, not too long ago it was impractical for individuals to have multiple phone numbers as it would be difficult to manage them. A lot of work managing contact information was done on paper and having a single short number was a tradeoff we chose for a better user experience to losing control and allowing anybody with our number to contact us. Though we have no control over who could contact us, we could stop spammers abusing this situation by registering with do-not-call registries and imposing external penalties on them if they violated these rules.

We have stopped using paper to store contact information a while back and even stopped entering individual numbers before placing a call. When we interact with others we select identities that are stored as names and the numbers that connect us have moved to the background. This is evident in the changes made to user interface of the new contact management application on Android smartphones as shown in in Fig 1.2. From a user experience perspective, we doubt users would care if all their contacts have one number or multiple as long as they are able to place a call or send a message by selecting their name from a list. They would
Figure 1.2: The new contact management application on Android smartphones [10] certainly appreciate the option to have control over their relationships provided the user experience is not degraded from what they are already accustomed to today. This control over relationships is especially important in an environment where imposing external penalties on spammers with do-not-call registries is ineffective [1] as their costs are too low and bad actors can easily hide behind state lines.

The amount of time we are going to spend in the future engaging with others using various digital tools is only going to increase from here and the best way forward is
to leverage the new environment we operate in and give users the option to have full control over the relationships they are entering into. These should not be transferrable to strangers and should allow both parties to have control over termination, instead of one requesting the other to stop contacting them. Having such tight control over every relationship may not seem necessary but by having unambiguous contracts for every piece of information being transferred, we increase the confidence users have on our digital tools and it allows them to share information knowing fully that the behavior is deterministic instead of messages randomly getting lost because of a spam filter.

1.1.2 Gestures

Chrome in an application is the group of elements like tool bars, scroll bars and buttons that surround the main content display on applications that typically run on personal computer operating systems. When display real estate became a scarce resource on touch devices like smartphones and tablets, chrome was the first thing that disappeared from applications and content occupied most of the screen. Since the most commonly used functions which were part of the chrome and easily accessible earlier have either disappeared into nested menus or occupy a tiny area, trying to operate these buttons on smartphones is a demonstration of Fitts’ law every time. To overcome this problem of invoking various functions without using buttons, developers have been steadily building a gesture vocabulary that users can perform on top of the content itself. The core insight behind these new ways to reuse swipe gestures for triggering specific functions is, there are instances where commonly used gestures become irrelevant and they can be repurposed to trigger a different function while still remaining intuitive for the user to perform.
The swipe down to refresh is one such gesture that has become ubiquitous in many mobile apps today. Swiping down when we are at the top of a list is meaningless and this was creatively used to replace the refresh button. Thus, a small button which was at the edge of a screen and left many users uncomfortable while they tried to access it is now readily accessible to them with a simple gesture.

Another important action that we repeatedly perform and is ripe for replacement provided the right gesture is available, is sharing selected items in a list. To achieve this we designed the tool to allow us to replace the current menu buttons with a pinch gesture that the user can use on top of selected items to share them with others.

1.1.3 Information

Another important piece when users share information with others is the information itself. Aspects like, who gets to see the information, how various parties that take part in transferring it use the information being transferred, and how much say users have over this process are very important.

The simplest function for a tool that handles information on behalf of users can do, is to take information from a sender and transfer it to a receiver. A tool that does only this simple function still has a lot of value to users which is demonstrated by the popularity and time spent by users on messaging applications. These tools would not even have to look at the information being transferred to provide this service. When current tools do not provide this guarantee, users resort to additional steps to encrypt data before transferring and decrypt it on the other end. Since only few users use encryption, they are more likely to be detected and flagged as suspicious for using it. A tool needs to embed encryption seamlessly into the default experience so
that maximum number of users can take advantage of it. In the past few years, we have been using our digital tools like email to get notified about critical information. Ensuring that only the right set of eyes can look into the data being transferred is very important as most of it is now passing through our inboxes rather than our mailboxes.

A while back, most of our incoming messages were only coming through one channel on email. But now with the increase in quantity we are segmenting our usage across multiple applications to manage our communication needs. But the people we are interacting with do not change across these applications. There could be an abstraction that developers can leverage using a common layer to ensure information is not locked in different application silos. Allowing multiple clients to transfer any type of data between users with a common API and providing other guarantees we discussed above could be one way to build such a layer.

1.2 Challenges and Issues

The most widely used platform to share information with others is email. There are two problems with email, the first one is spam and the second one is the inflexibility of what the protocol can carry to end users. The contents of an email are geared towards human consumption only and hence only email readers are built directly on top of the email transport layer. Though steps were taken by large providers to open up API access for an end user’s email, there is little flexibility for us to build other applications directly on top. This causes most applications to use the email infrastructure for user identification only and they roll out their own data transport infrastructure. An attempt at building a transport protocol that supports different
types of messages was made by Google Wave [2] but it ultimately failed to gain traction.

The other problem with applications bootstrapping off email is that a single address is being shared with multiple parties to reach the user which opens up possibilities of leaking the address to spammers which results in spam or unsolicited bulk email. Since the issue is with one email address being reused for many relationships, there were numerous attempts made to help users generate temporary user ids for every new relationship so that users can disable addresses when they leak and start receiving from on them. But we do not think this is solving the real problem but instead trying to find a workaround for a tool that does not support control over relationships natively. Since the messages transported on the Google Wave protocol were flexible, it tried to eliminate spam by introducing captchas, which definitely degraded the user experience; we doubt there are any users that like solving captchas.

Various machine learning models that are used to eliminate most spam [23] from a user’s email inbox are powered by signals coming from multiple users. Though most spam is caught, this approach makes it a blunt hammer that punches out the most problematic emails from everyone’s inbox but other email messages could be removed even if they are classified as spam only by a section of users. The fault is with the signals used to train these machine learning models and the behavior our existing tools encourage. When users are not interested in a message, it is easier for them to select the “This is spam” button in their email client instead of going through a few steps to sever the relationship with the sender. If we can give users options to sever relationships with senders completely with the click of a single button, we can encourage them to use that option and only send reliable signals to spam filters. An interesting side effect could be that bulk email senders clearly reach a set of interested
users and will not have to worry about email bounce rates.

To deal with the actual spammers, many solutions were proposed to shift the economics by charging a very small amount of money for every email sent so that legitimate users can afford it easily but spammers who send a very large number would not be able to. Moving money at such a wide scale is a bigger challenge than moving email and this is clearly not the right approach. The main reason spammers have it so easy today is because of very long validity of the lists of email ids they have harvested and the problem will partially be solved if we can invalidate a large portion of these lists of ids quickly. The resources required to maintain large valid lists itself could be a sufficient deterrent for spammers.

Another approach which many tools have leaned upon today is to ditch federated networks and have a central authority that controls all the aspects of a platform by monitoring usage and rate-limiting bad actors to reduce spam. But this would create an entity that has too much power in its hands that can be abused by imposing unfavorable terms unilaterally on others using the platform. Recently, platforms like Twitter have already used their super user status on their platforms by altering terms for third party developers unilaterally to suit their business goals.

1.3 Proposed Solution

We build a solution to address the challenges we discussed above by creating a prototype which allows users to start sharing information with a simple gesture, create new relationships effortlessly by scanning QR codes, allowing users to work seamlessly across multiple devices by being available on personal computers, smartphones and tablets. Users can share content using client applications on all devices and also be
notified immediately with push messages when others share content with them. Users will be able to quit relationships they are part of without the involvement of others in the relationship.

In Chapter 2 we take a look at the state of existing research and discuss existing solutions and research that have validated our approach. In Chapter 3 we go over the proposed solution to show how this thesis proposes to address the challenges we discussed. In Chapter 4 we look at the architecture of the complete prototype, the various modules that come together to form this tool. In Chapter 5 we look at the implementation details of each individual module and client applications. In Chapter 6 we evaluate the prototype with a peer review qualitatively using an After Scenario Questionnaire [16] with tasks that go through various functions of the prototype. We also use the Perceived Usefulness and Ease of Use Questionnaire [6] to look at whether metrics have improved between other tools that users already use and our prototype. A comparative analysis with several related tools and approaches is also included in Chapter 6. Chapter 7 concludes the thesis with a brief discussion on the various ways in which this work can be extended in the future.
CHAPTER 2
BACKGROUND AND RELATED WORK

In this chapter we touch upon various aspects of sharing information and we will look at how existing research shaped our solution. Learning curve is a very important element in the success of the adoption of a new tool. Wherever appropriate we have leveraged existing practices that are already widely in use and refactored them to work for our solution. This will result in a better experience for users because the learning curve to use our tool will be low. We split this chapter to cover the different parts that make up our tool and the relevant work in each area that helped us validate our solution or showed problems that can be improved upon.

2.1 Gesture

A list of items has become the default way to display information inside applications on mobile devices, there is no chrome and the list occupies the entire space from left to right on the screen unlike desktop applications. The difference in UI for the same application running in a browser and on a smartphone can be seen in Fig 2.1 and Fig 2.2. The functions we perform on a list of such items such as refresh or delete are pushed deep into menus or displayed as tiny buttons. This makes accessing these functions tough, especially if they are commonly used. Trying to find a place for a fixed number of functions in an application when screen sizes of the devices change is another challenge. There is a surprisingly large amount of optimization performed to display functions on the action bar of a phone because of this paucity of space, functions automatically go into a drop down menu when there is less space. Another widely used approach to solving this problem is to use intuitive gestures
instead of placing buttons to perform some of these widely used functions. Gestures also have an added advantage as they scale beautifully across multiple screen sizes without developers resorting to hacks. We will now look at three such gestures that are widely used across many applications.
Figure 2.3: Swipe down to refresh gesture in a smartphone application [29]

**Swipe down to refresh**

This gesture was first introduced by the mobile app Tweetie [3], this gesture is now ubiquitous on all apps that display a list of items and have a need to refresh it periodically to check for new items to display. When users are at the top of a list and would like to retrieve new items from the server, they swipe down on the list even when it is displaying the top most item. Since there is nothing else to do when a user swipes down at the top of a list, it is interpreted as a call to refresh with new items. An intuitive refresh icon animates to show that new items are being retrieved and they are displayed above the top most item in the list. This is shown in Fig 2.3.

**Swipe right to archive**

This gesture was first introduced in the mobile email client Mailbox and is now present on many other email clients. The user swipes right on an item in the list he wants to archive and remove from the current display. Since there is no other meaning for the swipe right gesture on a list item, it is unambiguously interpreted as an action to
Figure 2.4: Swipe right to archive an item gesture in a smartphone application [18]

archive the item. This is shown in Fig 2.4.

**Shake to send feedback**

Introduced in the Google Maps mobile app, this gesture is activated when a user shakes their device vigorously. When a shake is detected, it is interpreted as an intent to send the application developer feedback regarding glitches in user experience experienced by the user. A dialog box is overlaid on the current screen which the user can use to enter any feedback before sending it across to the developers of the application. This is shown in Fig 2.5.

These widely used established patterns show the need to replace common functions with intuitive gestures saving the users effort when they use them in their apps. The share button is a widely used function and users still reach into menus to use it. In our tool, we repurpose the pinch gesture to share selected items in a list.

In a previous era of computing devices, using multiple input devices like keyboard
and mouse were mandatory to transfer files from one computer to another. The paper Pick-and-Drop [25] outlines a technique where a single stylus is used to pick objects intuitively from one computer screen and deposit them onto another screen to transfer them digitally. SPARSH [20] removes the need for a stylus and outlines how objects can be transferred from one screen to another by touching both the screens with a finger. Deep Shot [4] outlines various ways to switch tasks between a phone and a computer using the camera on a phone. It demonstrates the use of a mobile phone camera to capture and process the current task running on a computer to migrate it to the smartphone and also to select the right computer to transfer the task running on a phone.
2.2 Pairing Users

Every time two users who are physically close want to connect on a digital platform, they end up exchanging each others identity information and enter this manually into their devices. We will discuss few attempts that tried to remove the need for users to type in usernames manually by leveraging other actions on the devices or using attributes like physical proximity or sensor information from the devices.

Bump was an application that users could use to share contact information by doing a physical bump of their fists holding phones in them. It used a combination of sensing simultaneous physical motion on devices and location information from GPS to identify the correct devices and exchange contact information.

WeChat users can shake their phone to look at a list of other users who shaked their phones at the same time. Users can use this feature when they meet others and want to exchange contact information. They simultaneously shake their devices and add the correct user by selecting them from the list of users displayed.

QR codes [22] are two-dimensional barcodes that are used to encode small snippets of textual information into a pattern of small white and black squares with three larger ones to orient the scanner processing the code. The size of the codes can vary and depends on the amount of information being and the error correction scheme being used. A sample QR code is shown in Fig 2.6. QR codes are very commonly used in advertisements to give readers a quick option to scan the code with their smartphones and visit a URL to gather more information.

Near ubiquity of a camera on all the devices users use today means we can use QR codes exclusively to exchange information between devices assuming that most users
will be able to use them comfortably with their devices. Currently, QR codes are already being used to remove the need to type userids when exchanging information or to pass around a session identifier from one device to another. WeChat is a instant messaging application for mobile phones and allows users to display a QR code on their phone with their username. This allows others to add a user’s account by scanning the QR code in. Facebook and Blackberry used QR codes to allow users to add new users to their friend list but they encoded the username into the QR code which was completely static.

Though we picked up a widely used best practice, we changed it slightly for our tool to serve a different purpose by embedding a temporary token in the QR code instead of a static token. Embedding temporary tokens in QR codes is also widely used but only to pair applications on devices that belong to a single user, like youtube app on a TV with a users phone, or google glass with a companion smartphone, or whatsapp web application with the smartphone app. We use temporary tokens to identify relationships between users and embed these long alphanumeric tokens in QR codes so that they can be quickly exchanged when users are in proximity.
From a user experience perspective there are many widely used applications like AliPay where QR codes play a major part in the user experience. This shows that users are comfortable using QR codes on their devices and we are using this to transmit relationship tokens.

2.3 Credit Card Tokenization

Credit card users face a dilemma when they share their card details with merchants while purchasing items from them. They always face a possibility where the card details could be revealed to thieves when the merchant’s systems get hacked. If the card is not blocked in time, the thieves can then use these card details to make purchases without the users consent. Replacing the card with a newer one is a hassle because there are other merchant relationships that depend on the same card number and all of them have to be updated with a new card number after a breach at a single merchant.

The various stakeholders in the credit card processing chain like merchants, card issuers and card processors have have come up with a solution to tackle this problem using tokenization [19]. The user submits the credit card number to a merchant who verifies the card details for validity and then requests the processor to issue a token. This token is then tied to this single user-merchant relationship and is stored as a substitute for the original card number. In the event of a database breach at this particular merchant, only the token that identifies the user-merchant relationship is revealed to hackers and only this is invalidated instead of the original card number. Users then need not replace their card or change any of their other merchant relationships. They would only have to recreate the relationship that was breached by
generating a new token for the merchant again. This system of using the original credit card number for authentication and generating multiple tokens to identify relationships and invalidating them individually without disturbing other relationships is certainly working for the credit card industry right now and there could certainly be a lesson for us to use it to improve the user experience for communication in general.

2.4 Control over Relationships in Federated Networks

Centralized networks have always afforded users control over their relationships because one entity controls all aspects of communication. Since we are trying to afford users the same amount of control in a federated network, we will take a look at attempts that others have made in currently widely used federated networks like email and instant messaging systems that send messages over XMPP to solve the same problem.

2.4.1 Email

Users lose control over the relationships they enter into when communicating with email because all communication is dependent on one id that is created during account setup and then shared with all relationships. Once we hand the id to another person to enter into a relationship, we have lost complete control over it. It can be shared with others without our permission, in a way forcing us to enter into relationships that we may not wish to, or stopping us from terminating the relationships we would want to be out of. We could technically terminate but that would be by putting a request to the sender and hoping they would remove our id by acting in good faith.
Users already use creative solutions to tackle this problem. To avoid getting picked up by spammers, they obfuscate the email id while posting it publicly on webpages. They click on unsubscribe links and hope that the sender removes them from mailing lists. They manage multiple accounts to segment personal email from other email they receive when they sign up at various websites and apps.

The authors of the paper XToken [27] describe a solution where users control their relationships by asking others to include special tokens they have exchanged before along with all the email they send. Though it solves the problem and gives users control over relationships without having to create temporary email ids, the authors of the paper admit that the solution has usability issues as it is a radical change from how users are accustomed to using email. This is mainly due to their efforts in trying to ensure compatibility with existing email infrastructure. These days, users have already moved away from email to other proprietary protocols for most of their communication needs. This gives us the confidence to try to build new tools instead of ensuring compatibility with older systems that are on the decline. By doing this, we can deliver users a more robust solution without degrading their user experience.

2.4.2 XMPP

XMPP [26] is another widely used protocol that allows users to send and receive instant messages in a federated network. It has a concept of users entering into relationships that they can control with others before sending messages. Instant messaging over XMPP is essentially a whitelist system where messages can only be exchanged between users only after they add each other to a list. Spammers do not reach the user unless they were approved to a friend list which eliminates a lot of
spam that plagues email systems. It is a whitelist based system and is not widely used for first contact with other users.

Although there is control over relationships when adding users to a buddy list, who can be removed by the user and will not get their messages again, a single user id is still being shared with everyone.
CHAPTER 3
PROPOSED SOLUTION

When a user shares information with others using a digital device like a phone or a personal computer, the entire process is roughly completed in two steps. In the first step the user collects information to be shared with others on a device. And in the second step, the user selects a list of recipients and then proceeds to share the collected information with them. Though there could be some amount of correlation between these two steps, they are mostly independent as the information being selected by a user to share with others could be shared with anybody.

We have used this insight to propose a solution that optimizes these two steps separately with different techniques. In one part we simplify collection of information by using a gesture and removing the need for the user to select multiple buttons. In another part we ensure the experience of the user connecting with others is enriched by giving them absolute control over all the relationships they enter into. We also allow multiple client applications to share this common user infrastructure to remove repetitive steps when sharing information from them at the same time.

3.1 Pinch Gesture

Now that touch devices have been a part of our lives for quite some time, we are starting to see some convergence in the gestures that developers are using across apps to call common functions. We discussed in the previous chapter how this helps users, but this also helps developers by removing the need to optimize placement of buttons for various screen sizes. In this section we show in detail how we are planning to use
Currently users go through multiple steps for each piece of information they share with other users. A user has to first select the information that he intends to share from a list, then select the share button to open a drop down menu, select a channel that will be used to send the information and finally select a person or create a group to send the information to. These steps have to be repeated multiple times if information resides in different applications even if it is shared with the same set of recipients. We try to condense the steps after selecting information to finish sharing that information with a single pinch gesture. The exponential increase in the number of steps a user has to perform is shown in Fig 3.1.

A user will do the normal long press and select one or more items in a list inside an application which could be a file manager, photo gallery or even items that are not files themselves like notes in a note taking application. Then after making a selection, the user will simply do a pinch gesture on the touch screen to send an intent to share this list. As we already know that pinch gesture is already used for
other things like changing the zoom level, it still would not interfere here because it is interpreted differently as the user has selected items in a list. The pinch gesture would be interpreted as a normal zoom gesture when items are not selected by the user. It would be interpreted as an intent to share when at least one item is in a selected state and the pinch gesture is performed on the list. Unlike the other gestures we discussed before which replaced actions that were complete after a button is released, sharing is only complete when the selected items reach the intended recipient. To do this, after the pinch gesture is done, we can show a pop-up menu to the user to select a channel to share the items with recipients. But to remove these extra steps we delved deeper and tried to complete sharing, including sending the selected items to recipients, with the pinch gesture itself. To achieve this, we built a common service that runs in the background on the device that all client applications on the device can hand over payloads to. The user will set an active relationship in this service to automatically send the selected items to the recipients on this relationship when the pinch gesture is performed in any client application on the device.

3.2 User Relationships

Sharing information with one or more users from a mobile device is currently good enough with applications already leveraging common features of the smartphone operating system like the user’s contacts list. But there are two things here that still hinder the user experience. The first one is immediate which happens when users are in the process of sharing information with others and another is over longer periods of time when dynamics of the user relationships that users are part of change in the real world.
Information that we could share with others resides across multiple applications on our devices and will continue to do so because each user will use have a different set of applications to cater to their needs. But the people we want to share the information with are the same across all the applications and the devices they use to receive information on does not change too. Applications have leveraged the contacts app in modern smartphone operating systems to allow users to select people to share information with a quick selection from the list of contacts on the device. But since the transfer of information is handled by each application independently, the selection of users has to be repeated across every application multiple times during the act of sharing information spread across multiple applications. Our solution uses a background service that allows the user to set one active relationship on the device to send information to. And then when a user performs a pinch gesture in multiple applications, the data selected is handed over to the background service on the device which sends it to the recipients on the currently active relationship. This allows the sharing process to be complete in one user action and removes multiple redundant steps which improves the user experience of the user during the sharing process.

The methods a system uses to manage relationships between users can have a big effect on the user experience in the long term. Currently when users want to form relationships with other users, both on federated networks like email or on centralized networks like Facebook or LinkedIn, they exchange their email ids to start a relationship.

We can look at the act of giving others our email ids as entering into relationships with them that they can use to send us messages. Not all relationships that we enter into with other users or organizations last a lifetime to deserve access to identification like our email id that lasts a lifetime. Either party should be able to terminate the
relationship they have entered into unilaterally without the consent or asking the other party to stop sending messages.

Several machine learning techniques that try to filter unsolicited messages including spam have not been very effective to solve the problem completely, i.e. stopping bad actors from sending them messages. The problem actually is not the algorithms not being robust, but rather the design allowing blatant leakages. We are trying to fix problems with our relationships, including the ones we entered into inadvertently due to problems with the underlying design, using machine learning at a layer above which just deals with delivering messages to various ids.

We aim to solve these problems by creating a system that can exchange messages between users without them exchanging long term identification details like email ids and phone numbers. The proposed architecture of this system is shown in Fig 3.2. Note that Figures 3.2, 3.3, 3.4 and 3.5 have been generated using the tool available at the website draw.io. When users want to start a relationship, they press a button in our application to create a unique token which is then shared with others to create a relationship. Recipients that wish to join the relationship then use the token to inform the server backend that they want to be added to that particular relationship. Once the server validates the token presented, it creates a relationship with all the users and they can then start sharing information payloads with each other using various client applications of their choice on their devices. The tokens are refreshed by the server whenever changes in the composition of the relationship take place to ensure the ones that leave do not have access to future information payloads that will be shared on the relationship. Once a participant wishes to leave and does not want to receive any more payloads being shared, they can terminate the relationship and the server will ensure they stop receiving messages from this relationship.
This was somewhat achieved in the past by the users creating temporary email ids to manage communication with untrustworthy parties. But this is not the right way to fix the problem as users on both ends have to generate temp ids of their own for every interaction which quickly becomes tough to manage, both for users and also for servers which were clearly designed with the idea that one user will use one email id. There is no point creating multiple user profiles for the same user to handle multiple relationships when we can segment them in one user profile itself.

A camera is one sensor that is now present in almost all devices we use today and we can leverage it along with QR codes to improve the user experience while exchanging information to create new relationships. Though we are replacing human readable userids with very long alphanumeric tokens that represent relationships in the system, by leveraging QR codes we can make the user experience even better by
Figure 3.3: Two users exchanging details to establish a new link between them removing the need to manually type in userids. Usually, they would have exchanged email ids but now one of them selects a button to create a QR code and the others can then use their devices to scan the QR code to join the relationship. The server then validates these details and adds them to the relationship. This is shown in Fig 3.3. Then everybody can start sharing information payloads with others present on the relationship without exchanging permanent details like email ids which they still use to login to the system. This is shown in Fig 3.4. At a later point, when a user wants to leave a relationship, they can do so by deleting it from their and the server will stop sending new payloads that are shared on the relationship in future. This is shown in Fig 3.5.

Though this concept can be applied to many scenarios, we wish to show that it is effective than the current methods in at least one scenario. We have created a solution that shows an improved user experience for users in a scenario where users want to share information with others they meet in person. We have created both a web application and mobile application to test the concept with multiple platforms. In this scenario, users who may or may not have met before but wish to share information
between them without exchanging details like emails ids can do so.

### 3.3 Information Payload

Another key goal for our solution was to create an architecture that can allow any type of information payload to pass through the server. For our solution, we have realized that since files are the current common standard that standardize along file extensions and allow multiple applications to work with a common data, we have
built a prototype that users can use to exchange files between mobile phones, tablets and personal computers. This helps users on a relationship to share a file and still use different applications individually to open them on different devices.

Though we limit our demonstration to transferring files between devices, the architecture would support any type of package as long as it is readable by a client application on the receivers end. If multiple client applications are willing to standardize and work with the same payload type, there is scope to use a richer namespace instead of limiting to files with extensions that are currently being used.

3.4 Privacy

Though improving user experience is very important, the security of the data should not be compromised while being handled by the server. The payloads are transferred over HTTPS between the devices and the backend. Hence, the security of data in our proposed solution is currently on par with existing services where data is only encrypted in transport between a user and the backend and there is scope for the administrator of the backend to access it.

We now outline a method where this architecture can be extended to improve privacy of data further if required. Currently a group of users already exchange a token out of band with others to form a relationship. They could exchange an extra token in the QR code that the server is not privy to. This can be used to derive private keys with a key derivation function on the client side and then use the keys to encrypt a payload before transferring it over the backend to other users in the relationship. Since the other users on the relationship already have the keys, they can decrypt the received payload before opening the contents on the device. With this
method, administrators of the server cannot look at the contents of the information being sent through. Whenever the composition of users changes in the relationship, the keys are then refreshed using the same encrypted channel already established but without notifying the user leaving the relationship with the new secret token. This will improve the user experience of encrypting data as it removes the need for the users to consciously go through additional steps when they want to encrypt and send data to others.

3.5 Usage Scenarios

Usage Scenario 1

John and Bill meet in a subway and start talking about their common interest in photography. They get very excited and promise to share with each other a few pictures they took recently. They want to share some pictures privately with each other but are apprehensive about giving their personal email ids. So instead, John opens the i80 application on his smartphone and creates a link. Bill opens his i80 application, scans the QR code displayed on John’s phone and they are now connected through a link. They depart the train station and go back to their homes. Later that night, John opens his Macbook, logs into the i80 web application, selects the pictures he wants to share and shares them with Bill through the link they established earlier. Bill is now notified in a few moments and he downloads the pictures to look at them. He also has a few pictures on his phone that he wants to share, so he selects them using the Pinch application, and does a pinch gesture to send them to John. John meanwhile leaves his desk and goes into the kitchen and his phone beeps with the notification and smiles looking at the excellent shots Bill just sent him.
Usage Scenario 2

A web retailer wants to launch a program where they share sales deals with interested users every few months. Even though the program shares good deals, very few customers sign up for it. The retailer guesses that may be customers do not want to give them an email id as they are a new player and probably lack trust. The retailer tries to reassure the customers about their top notch security practices in handling personal information and promise to never spam with too many emails, but they know deep down that they are not able to convince the customer reassuringly to share their email id. Without the trust of a recognizable brand, their promise sounds vague to even themselves, so they look for alternatives to solve the problem. So they decide to try i80 to handle their relationships, and use its API to integrate it into their systems. Now instead of a form asking customers to type an email id, they simply show a QR code generated for the user on their webpage. If the customer wishes to start a relationship with the retailer, they open their i80 app on a phone and scan the QR code displayed by the retailer. Customers realize that they are in control and can terminate the relationship whenever they like and also be reassured that their personal details are safe even if the retailer is hacked. And if the messages become overwhelming, they know they can end the relationship with the touch of a button.

3.6 Federated Design

Our solution that we have outlined here clearly looks like a centralized network where all the relationships that users enter into are managed by one backend. Due to time constraints we are not able to demonstrate a working solution to show that users can
Figure 3.6: Users communicating with others in a federated network.

enter into relationships that span multiple backends that are in different domains and managed independently like a federated network. However, we would like to show that this architecture is able to support relationships across multiple domains and still ensure users do not lose control over their relationships. Once terminated, the users decision is honored and messages do not reach them even if they are part of a different server.

The flow diagram in Fig 3.6 shows three servers A, B and C that can handle relationships for users independently. Let us assume Bill who is managed by A wants
to start a relationship with Bob who is at B and Alice who is at C. Since Bill creates the relationship at A, the token is marked with A’s domain and then passed to Bob and Alice through a QR code. When Bob and Alice accept the relationship, they send the token to their servers B and C, which recognize that the token originated at different domain A. So B and C send a confirmation to A. Then A marks that future communication on this relationship needs to be broadcast to all the local users part of it under its own server and also to B and C who then take care of distributing it to Bob and Alice. When Alice wants to share with others on the relationship, she sends a payload to C, which in turn forwards only to A. A looks at the relationship and realizes that it is intact the origin server for it and forwards the payload Alice sent to server B so that Bob can get it too. This clearly shows that users across servers can create relationships and ensure everyone can be reached when they send a payload.

Now, let us assume Alice decides to leave the relationship. She can leave it by disabling the token and notifying only the domain she has control over. Now even if A, being a different domain that Alice or C have no control over, continues to forward payloads to C, they will be blocked by C before they reach Alice as the relationship is marked as disabled. This shows that users which are not under the control of C still cannot send payloads to Alice because the relationship that was setup was invalidated at the recipient’s end which demonstrates that the concept of control over relationships can be extended to a federated setup where users are managed by multiple domains.
3.7 Next chapters

Sharing information is a large part of what we do daily with our devices and hence we could apply this to a large number of usage scenarios. But keeping the constraints of time in mind, we decided to create a solution that demonstrates the feasibility of the idea and shows clearly that the user experience is improved compared to existing tools by demonstrating it for one usage scenario. To demonstrate cross platform utility, we created a web application that can run on personal computers as a browser application and an Android application that runs on smartphones and tablets to cover a wide range of devices users use daily. To demonstrate the utility in terms of information being transferred by the system, we are transferring files between these devices. We have implemented the pinch gesture for the android application to pick files from a file manager app and send the files to other users on a relationship.
CHAPTER 4
SPECIFICATION AND DESIGN

4.1 Introduction

In the previous chapters we outlined a number of points that can improve various aspects of the information sharing experience when users use their digital devices. These ideas could be broadly applicable to multiple user scenarios but we focus on the design and implementation details of our solution for a particular use case that deals with sharing files. We briefly described this scenario previously but will elaborate further here and provide details of the design of this solution in this chapter.

4.2 Goals

The goals for our system are: allow users to send files with other users on a link; send push notifications to all the devices a user is currently logged in when new files are sent by others on a link; develop multiple client applications to support widely used operating systems and device form factors; allow users using touch enabled smartphones to use the Pinch gesture as an action to share files on a link; and allow users to display QR codes on their devices so that others can scan them and join the link in few quick steps.
4.3 Assumptions

We have made the following assumptions while designing the system: though the design of a federated network is feasible which can support links across multiple domains managed by different entities running the Pinch system, all our clients will connect to a single domain in our current implementation; clients are also designed to only contain files in the payloads that are transferred between various users in a link.

4.4 Features and Functionality

The sequence diagrams outline the main actions users perform with client applications and the interactions between clients and the backend.

Fig 4.1 shows the sequence of actions performed by the user and the clients to create a link and add other users to it. The diagram has two instances of a user, one user creates the link and another user who joins an existing link. It starts with
one user creating a link by performing the createLink action sending the server a POST request to the /links endpoint of the API. The server validates the request, and generates a uuid to create a link and returns the details of the link like linkid, link secret which are then stored in the client for future use. The user then displays the QR code on the screen and lets a new user who wants to join the link scan it with their device. The second user’s client then sends the server a PUT request to /links/linkid with the link secret requesting the server to add the user to the link. After validating the link secret, the server adds the user to the link and notifies both the users about the change to the link composition. The linkid stored on the clients of both the users can now be used by them to share files.

Fig 4.2 shows the sequence of actions that take place when users share files with each other on a link. A user first sets a link active from the list of links he is part of. Then to share a file, he selects a file and starts sharing either using a pinch gesture or
the upload button. The client then creates a payload with a new payload id and sends it to the server. After the upload is complete, the server looks up every other user on the link and sends a notification to the user on all the devices they are online. The users who are notified then proceed to download the files by selecting the notification that was sent to them.

4.5 Architecture

To implement the goals we outlined in the proposed solution above in a scalable manner we split the entire system into components like the backend, clients running on the users devices, and a messaging network as shown in Fig 4.3. The backend that acts as a central store of information for all users and the various clients they use to interact with others. There are two types of client applications that users can use to interact with the backend, a web application that runs in a browser and an android application that runs on android phones and tablets. The third major component is the messaging network that takes care of broadcasting a message generated by a user to others in the link in realtime.

4.5.1 Backend

The backend consists of a database that stores data about the various links in the system, uses a scalable object store to hold various payloads that the users shared with others in their links, and a messaging network to keep track of the devices that are online and push messages to users whenever new payloads are shared with them by others. Since we are planning to implement clients for multiple device platforms,
as well as allow flexibility for different clients to solve for different usage scenarios, we have consolidated the core business logic of the entire system in the backend behind an API server. This removes the need for clients to duplicate business logic and instead simply translate user actions into API calls to the server which then takes care of interacting with other parts in the backend. The backend also performs permission checks on objects to ensure users are able to retrieve only the objects they have control over. There is a separate web server that serves static assets of the web client that loads when users visit the website in a web browser.

**Payload Storage**

When a user tries to share data with others, it is first converted to a payload object by the client and then uploaded to the object store in the backend. After uploading it successfully, a key to the payload object is stored in the database and this is sent to other users on the link in a message. The users who receive this push message can then download the payload on any device they are logged into. All the users who are part of a link at a particular time have equal permissions over the payload objects.
uploaded by everyone to the link. And when a user leaves the link, they lose access
to all the payload objects that are part of it.

Database

The complete state of the system is stored inside a database. We picked a NoSQL key-
value database [24][14] and modeled all the access patterns of the system by storing
data across multiple tables. The lowest unit of storage in a table is an item which
can be retrieved using one of the values which is the primary key. This primary key
is unique across all items stored in one table. Other key-value pairs in the item can
be used to store various other attributes that are being stored in a specific table for.

The key item we are storing in a database are the links the users are part of and
all the keys to the payloads that were shared in each link. The most common queries
that our API exposes to clients are, retrieving all the links that a user belongs to,
all the users present in a single link, and all the payloads each link contains. We use
Access Pattern modeling to determine what information each table stores.

Each user can be part of multiple links and each link can have multiple users, so
we can construct a unique key by concatenating a userid and a linkid. So we have
one table that is used to store details relevant to a user’s link, like whether they are
an administrator, the devices that are part of, etc.

All the payloads belong to a link and these keys can easily be stored in one table,
the unique key will be a concatenation of the linkid and a payloadid.

Other information like user data, link specific information, have a straightforward
mapping, hence they are stored in individual tables whose primary keys are userid,
The API at a technical level hides the underlying complexity of the data stored in multiple tables and exposes a simple interface to clients where they can request data about entities at a higher level. It also acts provides an interface to all clients to request actions to be performed. In order to process each request the server looks up multiple database tables to create a response that is then sent to the user. The API server is a stateless component and looks up various database tables to retrieve information to create a response for every request that arrives from various clients.

Application server

Though the web client is designed to independently call the API and perform actions, the client application itself needs to be loaded in the browser when a user visits the website. The HTML and JavaScript assets that form the web application are served from an web server that listens to requests on a subdomain different from the API server. This client then interacts like other applications with the API server.

4.5.2 Clients

Web application

A web application that runs in the browser was implemented to allow users of all personal computer platforms like Windows, OS X and various Linux flavors to access
the system. The goal of the web application was to provide options to access the most common functionality like joining links, creating links, displaying the QR code, uploading files, downloading files shared by others on a link and receive realtime push notifications when new files are uploaded by other users. Once the assets are loaded, the user is prompted to login, and after the user logs in to the client, various API calls are made to fetch data relevant to the user. A dashboard is created and the user can then use the client to perform other actions.

**Android application**

The Android application supports both phone and tablet form factors. The application is split into two different parts, one deals with various link management functions and services that run in the background to let other applications use the service to share files. And another is a File Manager application that uses the exposed link management service to let users share files on the current active link with a pinch gesture.

**4.5.3 Payload Delivery**

With increasing realtime nature of our communications it is essential for our communication infrastructure to support realtime notifications of messages that are sent over a link. And users need to be notified when the application is in the background and users are not using it currently. Push notifications allow user's clients to receive messages right when they are published to the channel instead of polling manually at fixed intervals. However, clients still retain the flexibility to use the API endpoint payloadid/payloads to retrieve all the payloads that are present in a channel at any
point of time.

Our system deals with multiple users who in turn are part of multiple links with other users on the platform. From a publish perspective, every message that is created by a user needs to be broadcast to few other users on a link and from a subscribe perspective, every user receives messages being sent on multiple links he/she is part of. The best paradigm that elegantly handles this problem is the pub sub messaging paradigm where the publisher sends messages to topics and subscribers subscribe to the topics they are interested in. This allows both the publishers and subscribers to communicate in a scalable manner without tight coupling between them.

Each link is automatically assigned a channel in this messaging network. Various clients that a user uses to interact subscribe to this channel whenever they are online to receive messages broadcasted on this channel instantaneously. The backend server is the only publisher on the channel, when a user uploads a payload to the link, a message is created with the payload id by the server and broadcasted on the channel. All the clients of users that have subscribed to this channel and are currently online receive realtime notifications even when the user is not actively using the application. The client can download the payload when a user selects the notification. The clients ability to broadcast on this network is restricted to ensure all the messages are validated by the backend before sending them out to others that are part of a link.
In this chapter we will outline various details of how the applications and services that form i80 and Pinch are implemented and the various tools and services we used to build them.

5.1 i80

i80 has a backend with various services and applications running on servers in the cloud to store the entire state of the system and maintain connections with online clients to send and receive new updates. The various functions that can be performed by the backend are exposed to the i80 frontend clients running on user devices through REST API calls. Users can either use a frontend web application or an Android application to perform actions like creating links, joining links, deleting links and uploading files.

5.1.1 i80 Backend

Amazon Web Services (AWS) is a cloud infrastructure provider that provides a broad range of managed services which take the pain out of developers managing infrastructure to run services. All the operations to interact with services are conveniently wrapped in Software Development Kits (SDK) for various popular programming languages. Using these SDKs, developers can build scalable applications without going through the pains of managing infrastructure [5]. We have leveraged many such ser-
Figure 5.1: Various components in the i80 backend

Services offered by AWS to implement the architecture outlined in the previous chapter and they are shown in Fig 5.1.

Payload Storage

For payload storage, we use S3 which is an object storage service offered by AWS. It is used for storing the individual files that users share on links as objects that other users can download from whenever required. Each payload object is stored with a unique name that can be used to download later. The name is composed of two parts, the first part is the linkid of the link that the payload belongs to which is appended to the second part which is another id generated for every payload. These two parts make up the complete id for each individual payload. Using two different parts removes the possibility of collisions taking place when ids are being generated randomly for users on multiple links.

All the users part of a link, including the owner of the link, have equal access rights over all the payloads uploaded to the link. Hence we do not have methods to assign different sets of permission for each object among users part of a single link.
To save resources, we have used one particular optimization to allow clients to directly upload payloads and retrieve them from the object store directly. The actual payload files do not pass through the API server, though they are tracked in the database with various API calls. The client requests a signed url with an API call and then directly uploads the file to the storage service bypassing the API server. When the upload is complete, the client makes another API call to notify the API server that a file is now present in the object store for the URL sent earlier.

The API server then proceeds to notify the other users present on the link about a new file that is shared with them. When a client wishes to retrieve a payload from the backend, an API request is sent and a signed URL is retrieved from the response. This signed URL can be used to directly retrieve the payload from the object store. Though clients interact with the object store directly, they do so only with URLs that are signed prior with a secret key that only the server has control over. Hence, even with the knowledge of a complete payload id, for example trying to access a payload after they have left a link, they wouldn’t be able to retrieve it from the object store as a signed URL cannot be constructed without the knowledge of the secret key.

Database

For storing state of all the users interacting on i80, we use DynamoDB which is a managed database offered by AWS. We chose this database because it provides the right balance between allowing us to flexibly design the database model and the types of queries we can run to retrieve data to serve the various requests delivered by the API server.

DynamoDB is a key-value store, which unlike relational databases which allow us
to query with any column of data, allows us to query using the unique primary key value of an item in a table and a limited number of secondary indexes that we can build for certain other keys in the item. The amount of data we have to go through to serve a request is crucial to performance. Too much data to parse through to serve a subset for a request affects performance, especially if it is for commonly used queries. Since we typically need to run queries against only one or two keys per item, this is a perfect fit for us. Apart from the keys that are used to query for data, the rest of the keys in the item are flexible. This allows us to experiment with new features by adding additional keys to the items without costly refactoring of the database schema.

The most common queries we perform from our API server are, getting a list of links a single user is part of, getting a list of users part of a single link, a user setting which of his personal devices need to be notified for new messages for a single link, getting a list of payloads that were uploaded to a link, and getting various details of the devices that a user accesses the data from.

A userid and a linkid when combined together can form a unique primary key in a table. Hence we use these two values together in a table as a primary key and store other relevant details in other keys. Since two different queries are run on this table, we build a secondary index by reversing both the ids and using linkid and user id to query the same details but by using a single link id. The payloads uploaded by all users in a link are not tied to any particular user and only to links. So the payload ids are appended to a linkid to form the primary key of a new table. This table stores other details like, the type of the payload, etc. Another query that is performed is accessing details of individual devices that a user uses to access the system. These details are again separated from the main table for building efficient queries. Other
user specific details like the subscription key are not tied to any particular link and are common for all the links, hence there are stored in a separate table with the userid as the primary key. A pattern we used to model this data is to avoid duplicating any item details in multiple items of a table.

The complexity of our data model, keeping the constraints of the underlying database in mind which is split across multiple tables, is not exposed to the clients as the data from multiple tables is composed together when the requests are served from the API. The next section discusses how data from these tables are composed together when serving the requests coming from various clients.

**Web Server**

A user visits the URL, www.i80.io, in a browser to open the i80 web application and interact with other users. We use a server instance on AWS EC2 that runs a Nginx web server to handle HTTP requests and load the web application in the browser. Every HTTP request that comes from a front end client is routed to Nginx. The static files (HTML, CSS, JavaScript and images) of the web application are then sent to the browser. The implementation of the web application is discussed in the i80 Frontend section later in this chapter.

**API Server**

Frontend clients modify state in the backend by making various API requests on the api.i80.io subdomain. These requests are forwarded to the NodeJS server running on the same server instance in AWS EC2 we mentioned in the previous section.
There are many common tasks specific an API server needs to perform like manage user authentication, etc. To perform these for our API server, we use an open source server-side web framework called Hapi that runs on NodeJS and allows us to focus only on logic required for each API request. To ensure scalability of the backend, we made the API server stateless and the clients need not maintain persistent connections for a long time, only enough time to serve a single API request. It is the responsibility of clients to maintain their current state and include necessary data in their requests, Hapi processes these requests with the necessary business logic and then interacts with the other services like the database, object store and the messaging network. Once all the operations for a request are completed in these services, a response is created and sent to the client to complete the request. The persistent connections that the clients maintain are with the messaging network. Since this is the only service used in the system that is not managed on an infrastructure level by AWS, a server crash would limit the failure to the currently outstanding requests which the clients can re-send to a different server instance. This ensures that the API server is both scalable and robust to handle failures.

The various API requests that the server allows the clients to perform follow RESTful design principles [7]. The various abstract entities like links, users, payloads and devices are exposed through various endpoints. Existing entities can be retrieved with GET, new ones can be created with POST, existing ones can be updated with PUT and deleted with a DELETE HTTP request. The data that is served form these endpoints is composed by the API server after querying various tables stored in DynamoDB.

Although there is a possibility for the clients to directly interact with the database, we achieve two things by making them issue API calls to perform actions instead of
### Table 5.1: Available API calls for Link Management on i80

<table>
<thead>
<tr>
<th>Request Type</th>
<th>Endpoint</th>
<th>Form Data</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>/links</td>
<td>linkname</td>
<td>Create a new link and return details</td>
</tr>
<tr>
<td>PUT</td>
<td>/links/{linkid}</td>
<td>linkname, linksecret</td>
<td>Join an existing link with given inputs and return details</td>
</tr>
<tr>
<td>GET</td>
<td>/links</td>
<td></td>
<td>Return a list of all links the user is part of</td>
</tr>
<tr>
<td>GET</td>
<td>/links/{linkid}</td>
<td></td>
<td>Get the details of a particular link</td>
</tr>
<tr>
<td>DELETE</td>
<td>/links/{linkid}</td>
<td></td>
<td>Remove a user from a link to stop new payload messages</td>
</tr>
<tr>
<td>PATCH</td>
<td>/links/{linkid}</td>
<td>updates</td>
<td>Update the link for a user with the modifications supplied</td>
</tr>
</tbody>
</table>

* substitute `{linkid}` with the id of the link.

The various API calls that we have implemented and exposed to the front-end clients and the functionality they implement is outlined in the four tables Table 5.1, Table 5.2, Table 5.3 and Table 5.4.
Realtime push messages when a user uploads a payload are not sent directly to client devices by the API server, instead they are routed to a message broker, in our case is an independent cloud service called PubNub. PubNub is a cloud-based Publish/Subscribe [8] provider which allows developers to manage channels that our i80 API server publishes messages to on behalf of clients and allows client devices to subscribe and receive messages in realtime when they are published. We can de-
<table>
<thead>
<tr>
<th>Request Type</th>
<th>Endpoint</th>
<th>Form Data</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/links/{linkid}</td>
<td></td>
<td>Return the list of payloads uploaded to a link</td>
</tr>
<tr>
<td></td>
<td>/payloads</td>
<td></td>
<td></td>
</tr>
<tr>
<td>POST</td>
<td>/links/{linkid}</td>
<td>name, description</td>
<td>Create a new Payload and return signed URL for the upload location</td>
</tr>
<tr>
<td></td>
<td>/payloads</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUT</td>
<td>/links/{linkid}</td>
<td>file</td>
<td>Update existing Payload details, ex: Modify item when a Payload has been successfully uploaded to Amazon S3</td>
</tr>
<tr>
<td></td>
<td>/payloads</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>/{payloadid}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>/links/{linkid}</td>
<td></td>
<td>Retrieve details of a single payload</td>
</tr>
<tr>
<td></td>
<td>/payloads</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>/{payloadid}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* substitute {linkid} with the id of the link.
* substitute {payloadid} with the id of the payload.

Table 5.4: Available API calls for Payload Management on i80

fine access control permissions for user at the channel level to ensure only the users with right permissions can subscribe to the channels they belong to. Every new link created in i80 has a corresponding channel in PubNub that the i80 API server will publish messages to and the clients will subscribe and receive messages from.

Every user is assigned a new subscription key when the account is created in i80. When a user joins a link, this key is granted access to the corresponding channel so that the user’s devices are able to receive messages whenever they are published by the server. When a user deletes a link, the personal subscription key is revoked access to the channel which ensures the user will not receive any messages broadcast on the channel in future. Though clients can publish to these channels directly, we removed
the ability to do so and only allow them to subscribe only. Every time a new link is created, as a part of the creation process, the API server generates a publish key and grants publish access to the channel. This is stored in a database table and none of the front end clients receive access to this key.

When a user joins an existing link which has a channel in PubNub already, the API server retrieves the user’s secret access key and assigns it with rights to subscribe to the channel. Once subscribed the user receives notifications from the server when new payloads are uploaded. When a user deletes a link, the user key used to get access to a channel is revoked access.

When a new link is created, the server also creates a channel in PubNub and creates and assigns a new key that has publish rights to the channel. This key is stored in the database under the link details table. It is retrieved by the server when a user successfully uploads a new payload and a notification has to be sent to other users who are online and are listening on the channel for updates.

5.1.2 i80 Mobile Frontend

The Android application handles multiple tasks to enable the user to manage the links they are part of and send/receive messages to others. One module deals with managing the links, allowing the user to join new ones or quit existing ones. This module sets the active link and other applications package and handover a payload and this module transfers it to the server. Once the user logs in, it retrieves existing links and the keys required to subscribe to notifications on the PubNub channel. It can be used to manually retrieve the payloads shared on the link. Another module is a background notification service that maintains an active connection to the PubNub
Figure 5.2: i80 and Pinch Mobile applications

messaging network and listens to broadcast messages, and sends notifications to the user when messages arrive. The background notification service that listens to PubNub messages published to the channels that the user is part of. It generates a device notification to inform the user about new payload uploads to a channel. When the user selects this notification, the payload is retrieved and downloaded to the device.

Both i80 and Pinch applications are visible in Fig 5.2.

After the i80 application is opened, the login screen of the app is displayed. This is shown in Fig 5.3.

After a successful login, the list of links a user is part of is populated. The first screen displayed has options for the user to type a link name and can create a new link or join one by scanning the QR code. This is shown in Fig 5.4.
When another user shares a new payload, a notification is displayed on the phone with the details of the payload. This is shown in Fig 5.5.

5.1.3 i80 Web Frontend

The main goal of the web application is to allow users on personal computers to access payloads being sent over links. The web client implements the most common actions that are needed for a user to create a link, join a link, upload a file and to retrieve all the payloads shared on the link. To achieve this, two pages are implemented.

The user can then start interacting with earlier links or create new ones, or upload new payloads to any link they are part of. We have used the AngularJS web framework to implement the web application.
The first page that is loaded when a user visits the URL is the login page as shown in Fig 5.6.

Upon successful login, a user’s data is retrieved using a few API calls and the dashboard page is populated with the details. This is shown in Fig 5.7. The dashboard has many components that give user the ability to manage links, upload files and receive messages. The Current Active link message displays the active link to which the files will be uploaded. The sidebar menu has buttons to Create and Join new links and a list of links a user is part of. Each linkname also has a delete button that a user can use to stop receiving messages from a link. The file upload box is used to select the file and upload to the active link. The QR code is refreshed whenever the active link changes to reflect the correct linkid and linksecret when other users scan to join. The messages box displays the real-time messages being received on all the
links a user is currently part of. The payloads box shows the entire list of payloads that have been uploaded to the currently active link.

When the user selects the Create Link button, a dialog box is displayed to give it a name and select the Create button. This is shown in Fig 5.8.

When the user selects the Join Link button, a dialog box is displayed to give it a
Figure 5.7: Main dashboard page

Figure 5.8: Dialog box with form for Create Link
5.2 Pinch

We have built a file manager application called Pinch to demonstrate how an application can interact with the i80 modules to send/receive files on i80 as well as to show a demonstration of the pinch gesture to share files with the links. The gesture is used to package the selected information in an application and hand it over to the active link currently on the device. A scale gesture detector on a gesture overlay [17] is used to detect the pinch. This detector is activated over the activity that renders the list of items only when at least a single item from the list is selected by the user. When
the scaling factor of the two finger pinch gesture is less than 0, then the method is called to read the path of the currently selected file. This file is then handed over to the i80 service to be uploaded to the server on the current active link on the device. Once the file is uploaded, the database record is updated and a message is generated and published on the corresponding PubNub channel which is then received by all the users part of the link and currently online.

Before a file can be shared, an active link has to be set for the whole device. This is done by selecting any link from the list of links in the i80 application as shown in Fig 5.10.

Once selected, a permanent notification is visible in the notification drawer as shown in Fig 5.11.
To select a file, the Pinch application is opened as shown in Fig 5.10.

A long press on any file selects the file. The highlight color changes when a file is in a selected state as shown in Fig 5.13.

Once a file is in selected state, it can be shared with users part of the current active link on the phone with a simple pinch gesture anywhere on the display. This is shown in Fig 5.14.
Figure 5.12: Browse folders in the Pinch application

Figure 5.13: Select file and do pinch gesture
Figure 5.14: File uploaded toast notification
In this chapter we use two methods to evaluate both the implementations of i80 and Pinch together as a single system that helps users have control over links with others and share files using these links. There is no substitute to conducting a formal user study and evaluate the effectiveness of the user interfaces developed in solving the goals they are designed for. However, since we were constrained by time, we have gathered feedback on the high fidelity prototypes of i80 and Pinch from fellow software developers [9]. This group of users are both well versed in the development process of software applications as well as use a variety of applications for similar use cases in their personal and professional lives. The experimental evaluation section outlines the test material and setup used to obtain the feedback and then we discuss the results. The comparative evaluation section contains a brief analysis of various features that i80 and Pinch offer with relation to various commercially available applications that solve similar use cases.

6.1 Experimental Evaluation

6.1.1 Test Plan and Materials

Six computer programmers from the investigator’s peer graduate student group took part in the session and shared their feedback by filling questionnaires and making notes. We have used the Google Chrome browser on a Macbook to run the i80 web application, and two mobile devices, an Nexus Android smartphone and a Nexus Android tablet, to run the i80 mobile applications. The various backend services were
also running to enable the client applications running on various devices to interact with each other. For the purposes of simulating real world information sharing between users in the test, we have logged in three different users A, B and C into the personal computer, smartphone and tablet respectively.

Before beginning the test and handing over the questionnaire material, the investigator briefly explained the user interfaces of both these applications to familiarize the test participant. The various scenarios outlined in the study asked test participants to switch between these devices as necessary to perform the tasks.

The test participants completed five different tasks in 30 minute slots that showcased both the overall functionality as well as highlighted the strengths that i80 and Pinch offer like control over relationships and the intuitive pinch gesture to share files. To ensure the test participants relate to the tools they use in their daily lives, every task also had a hint to make the participant recollect from personal experiences about how they performed a similar task using other applications and then compare the experience with i80 and Pinch.

An After Scenario Questionnaire [16] was administered after every individual task and participants were encouraged to make notes to give comments to the investigator. ASQ was developed by Jim Lewis in 1991. The ratings in ASQ are completed on a seven-point scale from Strongly Agree to Strongly Disagree and is used to judge the tester’s perception of effectiveness, efficiency, and satisfaction in completing the task. A sample ASQ questionnaire used by the test participants is shown in Fig 6.1.

The five tasks that the users performed along with the hints are given in Table 6.1.

After the participants completed all the tasks above, they were given the Perceived Usefulness and Ease of Use [6] questionnaire to evaluate both the usefulness and ease
Please rate the usability of the system for the previous scenario.

Try to respond to all the items.

For items that are not applicable, use: NA

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>NA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall, I am satisfied with the ease of completing the tasks in this scenario</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>strongly disagree</td>
</tr>
<tr>
<td>Overall, I am satisfied with the amount of time it took to complete the tasks in this scenario</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>strongly agree</td>
</tr>
<tr>
<td>Overall, I am satisfied with the support information (text, etc) when completing the tasks</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>strongly agree</td>
</tr>
</tbody>
</table>

Figure 6.1: After scenario questionnaire

of use of the complete system. They were also asked to list three negative aspects and three positive aspects of the system. This questionnaire was used to allow the test participants to focus on their entire experience using the system instead of specific tasks. PUEU was developed by Davis, F. D. in 1989 and a sample questionnaire used by the participants is provided in Fig 6.2, Fig 6.3 and Fig 6.4. Throughout the test, they were also encouraged to write down notes about things they felt like sharing with the investigator and some of them are listed at the end of this section.
<table>
<thead>
<tr>
<th>Task Description</th>
<th>Task Hint</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Creating a new link on web app and make another user join using the phone</td>
<td>Think about the ways you exchange contact information with other users.</td>
</tr>
<tr>
<td>2 Share a file on a link using the webapp, check push notification on the phone</td>
<td>Think about many ways in which you upload files online and then share them with others</td>
</tr>
<tr>
<td>3 Set an active link, Share a file on the phone using the pinch gesture in the Pinch app</td>
<td>Think about selecting an item on a phone, selecting the share button, entering or selecting recipient contact information and sharing</td>
</tr>
<tr>
<td>4 Check a real-time message received on the webapp and download file</td>
<td>Think about receiving a realtime message notification</td>
</tr>
<tr>
<td>5 Create a link, Join it from the phone, Use Pinch to share a file and check it is received, Delete the link in webapp, Share another file from a phone and verify that no message is received now</td>
<td>Think about trying to stop someone from sending you emails or messages, your traumatic experiences battling spam trying to unsubscribe from mailing lists!</td>
</tr>
</tbody>
</table>

Table 6.1: Tasks performed by participants on i80 and Pinch

6.1.2 Results

ASQ Results

Every question on the ASQ asked the participant to choose one of the 7 levels between Strongly Agree and Strongly Disagree and each level is worth 1 point. Since we were trying to find the effectiveness of the system across individual tasks as compared to other tools that the participants use, we have computed an average and the standard deviation for all participants across each task. The results are summarized in Table 6.2. The results ranged from a low of 5.83 to a high of 7.00 (on a scale of 7) across all the tasks. Task 5 had the highest number of steps for the test participants.
Please rate the usefulness and ease of use of the system.

Try to respond to all the items.

For items that are not applicable, use: NA

**Perceived Usefulness**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>NA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using the system in my job would enable me to accomplish tasks more quickly</td>
<td>unlikely</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>likely</td>
</tr>
<tr>
<td>Using the system would improve my job performance</td>
<td>unlikely</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>likely</td>
</tr>
<tr>
<td>Using the system in my job would increase my productivity</td>
<td>unlikely</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>likely</td>
</tr>
<tr>
<td>Using the system would enhance my effectiveness on the job</td>
<td>unlikely</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>likely</td>
</tr>
<tr>
<td>Using the system would make it easier to do my job</td>
<td>unlikely</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>likely</td>
</tr>
<tr>
<td>I would find the system useful in my job</td>
<td>unlikely</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>likely</td>
</tr>
</tbody>
</table>

Figure 6.2: Perceived usefulness part of the PUEU questionnaire

to perform and received the lowest scores among all the tasks. This coupled with the fact that Question 3 which asks the participants to rate the availability of support information shows that there is lot of scope for improvement in both documentation as well as helpful cues like tooltips and progress indicators to serve as a better guide to users interacting with the system. The comments left by users while performing the tasks are summarized in the Observations section later in this chapter.
## Perceived Ease of Use

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>NA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning to operate the system would be easy for me</td>
<td>unlikely</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>likely</td>
</tr>
<tr>
<td>I would find it easy to get the system to do what I want it to do</td>
<td>unlikely</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>likely</td>
</tr>
<tr>
<td>My interaction with the system would be clear and understandable</td>
<td>unlikely</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>likely</td>
</tr>
<tr>
<td>I would find the system to be flexible to interact with</td>
<td>unlikely</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>likely</td>
</tr>
<tr>
<td>It would be easy for me to become skillful at using the system</td>
<td>unlikely</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>likely</td>
</tr>
<tr>
<td>I would find the system easy to use</td>
<td>unlikely</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>likely</td>
</tr>
</tbody>
</table>

Figure 6.3: Perceived ease of use part of the PUEU questionnaire

List the most **negative** aspect(s):
1.
2.
3.

List the most **positive** aspect(s):
1.
2.
3.

Figure 6.4: List negative and positive aspects part of the PUEU questionnaire
<table>
<thead>
<tr>
<th>Task</th>
<th>Q1 - AVG</th>
<th>Q1 - SD</th>
<th>Q2 - AVG</th>
<th>Q2 - SD</th>
<th>Q3 - AVG</th>
<th>Q3 - SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1</td>
<td>6.66</td>
<td>0.74</td>
<td>6.00</td>
<td>0.74</td>
<td>6.00</td>
<td>0.57</td>
</tr>
<tr>
<td>Task 2</td>
<td>6.50</td>
<td>0.50</td>
<td>6.33</td>
<td>0.74</td>
<td>6.33</td>
<td>0.74</td>
</tr>
<tr>
<td>Task 3</td>
<td>6.83</td>
<td>0.37</td>
<td>6.83</td>
<td>0.37</td>
<td>6.33</td>
<td>0.47</td>
</tr>
<tr>
<td>Task 4</td>
<td>6.83</td>
<td>0.37</td>
<td>7.00</td>
<td>0.00</td>
<td>6.66</td>
<td>0.47</td>
</tr>
<tr>
<td>Task 5</td>
<td>6.83</td>
<td>0.74</td>
<td>6.67</td>
<td>0.47</td>
<td>5.83</td>
<td>0.68</td>
</tr>
</tbody>
</table>

Table 6.2: After scenario questionnaire results

**PUEU Results**

The PUEU questionnaire was divided into three sections, Perceived usefulness, Perceived ease of use and a comments section to list the negative and positive aspects of the entire system. We have used PUEU to gauge the overall effectiveness of the system instead of individual tasks. The first two questionnaire sections had six questions each and the participant has to choose one of the 7 levels between Strongly Agree to Strongly Disagree and each level is worth 1 point with the total score of 84. The PUEU results are summarized in Table 6.3. The average across all test participants was 75.83 (out of a total of 84) and the median score was 75.50.

**Observations**

The negative aspects listed by test participants in the questionnaire are:

- The lack of written documentation and helpful cues in the user interface like tool tips made it difficult to perform actions when stuck.
<table>
<thead>
<tr>
<th>Participant</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Participant 1</td>
<td>75</td>
</tr>
<tr>
<td>Participant 2</td>
<td>69</td>
</tr>
<tr>
<td>Participant 3</td>
<td>80</td>
</tr>
<tr>
<td>Participant 4</td>
<td>80</td>
</tr>
<tr>
<td>Participant 5</td>
<td>75</td>
</tr>
<tr>
<td>Participant 6</td>
<td>76</td>
</tr>
</tbody>
</table>

**Average** 75.83

**Median** 75.50

Table 6.3: Perceived usefulness and ease of use questionnaire results

- When there are more than one user in a link, both applications currently do not surface the names of the individual usernames that are part of a link.

- Since users who are part of a link can choose to leave it on their own, every user on a link should also be intimated with a message whenever one of them from a group of users part of a link leaves it to avoid confusion.

- The use of QR codes saves time for users to join a new link but it only convenient when both users who want to start a link are in the same physical location. There should be alternate methods to share the linkid and linksecret through a URL so that users far apart can also join the same link easily.

- In current applications that use the Share button, although there are a lot of steps, two fingers are not needed to complete sharing, but they are needed to do a pinch gesture which may not be an option in some situations.

The positive aspects listed by test participants in the questionnaire are:

- They liked the idea of having control over relationships very much and found
sharing with the pinch gesture easy and faster to use when compared to traditional tools.

- They found that there is a very less chance of identity being leaked with i80 as information like personal email ids and phone numbers are not required to start sharing files.
- They think it will be easy to reduce unwanted messages with better control over relationships.
- They liked that they were able to use it on multiple types of devices.
- They felt the Pinch gesture is unique in the use case it was being applied to.
- They identified that i80 has potential in sharing not just files but information from diverse set of other applications.

Few other comments left by test participants are:

- Navigating through the mobile app interface to create and join links is confusing.
- There should be a progress bar in both the web and android application when files are being uploaded to indicate both the status and the completion of the upload.
- Currently there are no options to delete a single file uploaded to the link when browsing through the list of payloads present in a link.
- Instead of selecting activelink before starting the sharing process on a mobile device, a list of links could be displayed right after pinch gesture. This will still reduce the number of steps and be intuitive.
- It could be hard to do, but link management should be integrated into the Android settings menu as it looks like a system level service to multiple applications.
6.2 Comparative Evaluation

There are many services available today that users can use to share files with others using applications on their browsers and smartphones. We select types of services to evaluate i80 with by comparing various features. Both the comparison points and the number of services we are comparing are not exhaustive but can serve to identify some major differences. Additionally, we would like to note that the three classes of services used for comparison have other powerful capabilities and features that our proposed solution does not have and were not included in the study. These capabilities are particularly embodied in specific applications belonging to each type.

File storage and sharing services allow users to automatically backup certain folders on their computers online and share them with others. They also allow users to sync them with their other personal devices and also among multiple users if the folder is shared among them.

Messaging services allow users to use their smartphones to send and receive text messages among friends. They also support messaging between groups of users and allows them to send media like pictures and videos.

We compare various features that we have implemented in i80 and Pinch along with Email and these two classes of services in Table 6.4.
Table 6.4: Comparative evaluation of features of i80, Messaging services, File sharing services and Email

<table>
<thead>
<tr>
<th>Feature</th>
<th>i80</th>
<th>Messaging services</th>
<th>File sharing services</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control over relationships</td>
<td>Yes, absolute control</td>
<td>Partial, can block users from sending messages</td>
<td>NA</td>
<td>No</td>
</tr>
<tr>
<td>Multiple payload types</td>
<td>Yes</td>
<td>No, limited to text and media</td>
<td>No, limited to files</td>
<td>No, limited to text and files</td>
</tr>
<tr>
<td>OS Platform support</td>
<td>Web and Android</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>User groups support</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No group construct available</td>
</tr>
<tr>
<td>Realtime notifications</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Not natively but possible</td>
</tr>
<tr>
<td>Archiving messages</td>
<td>Yes</td>
<td>No. Backed up at client level</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Encryption</td>
<td>User to Server</td>
<td>User to User except for groups</td>
<td>User to Server</td>
<td>User to Server</td>
</tr>
<tr>
<td>User Contacts</td>
<td>Links</td>
<td>Phone Contacts</td>
<td>Email ids</td>
<td>Email ids</td>
</tr>
<tr>
<td>Effectiveness at tackling spam</td>
<td>Very High</td>
<td>Very High</td>
<td>NA</td>
<td>Low</td>
</tr>
</tbody>
</table>

*Continued on next page*
We can also compare the differences in actions users have to perform to share files in these applications and i80 and Pinch. The number of actions that a user has to perform on a smartphone to share a file to another user is a pretty straightforward indicator of the speed an efficiency of various techniques that can be used to do it. The potential gains from using i80 and Pinch compared to existing approaches are summarized in Fig 6.5.

The difference in number of steps between these two approaches can be seen in the step diagram. We have classified all the actions into three categories, selection actions that are performed to select the content that is going to be shared, share actions that are performed to start the sharing process and the user typically selects a provider, destination actions to select the contact information of the target user who will receive the shared content. The step diagram shows the sequence in which a user performs these classes of actions and the gains achieved when i80 and Pinch are used to share information from three different applications with the same recipient. The

<table>
<thead>
<tr>
<th>Feature</th>
<th>i80</th>
<th>Messaging services</th>
<th>File sharing services</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple client applications</td>
<td>Yes</td>
<td>No</td>
<td>NA</td>
<td>Yes</td>
</tr>
<tr>
<td>Support for building services on top</td>
<td>Yes, can build new clients with the API</td>
<td>No support</td>
<td>API allows new applications to store files</td>
<td>Some support is available</td>
</tr>
</tbody>
</table>

We can also compare the differences in actions users have to perform to share files in these applications and i80 and Pinch. The number of actions that a user has to perform on a smartphone to share a file to another user is a pretty straightforward indicator of the speed an efficiency of various techniques that can be used to do it. The potential gains from using i80 and Pinch compared to existing approaches are summarized in Fig 6.5.

The difference in number of steps between these two approaches can be seen in the step diagram. We have classified all the actions into three categories, selection actions that are performed to select the content that is going to be shared, share actions that are performed to start the sharing process and the user typically selects a provider, destination actions to select the contact information of the target user who will receive the shared content. The step diagram shows the sequence in which a user performs these classes of actions and the gains achieved when i80 and Pinch are used to share information from three different applications with the same recipient. The
destination setup actions are not repeated multiple times and instead performed once at the beginning, the item selection actions remain the same and two share actions are condensed into one action which is an intuitive pinch gesture.
CHAPTER 7

FUTURE WORK AND CONCLUSIONS

Future work for i80 and Pinch includes:

- Expanding backend software to provide support for a federated network, with multiple servers;
- Extending the payloads, which currently carry only files, to other items, e.g. notes, reminders, and text messages. This will allow the proposed system to support other types of applications, such as messaging clients and note sharing;
- Enhancing the security aspects of sharing information by transferring a symmetric encryption key when users initially form a link using an out-of-band channel that the server backend has no control over. This will allow data to be more secure and will provide increased privacy to users;
- By standardizing the payload types users on different ends of the link can run different applications that work with the same type of data. That will generalize the use of the proposed approach and infrastructure.

Other future work tasks we can focus on are:

- Testing the proposed system with more users;
- Developing and conducting formal usability studies;
- Implementing Pinch on different platforms such as the iOS.

In conclusion, the main strengths and contributions of the proposed approach and related infrastructure include:
• Taking advantage of the pinch gesture on touchscreen devices to allow users to share files. This is an original, novel idea that adds to the repertoire of already used gestures for triggering functions on touchscreen devices;

• Demonstrating that users can directly control the relationships established with other users over a federated network;

• Designing and implementing i80, a communication and sharing approach that allows users to be in control of their short term relationships without giving others permanent credentials like email ids and phone numbers that are valid for a longer term;

• Designing and implementing Pinch, an Android mobile application for sharing files with other users on i80;

• Testing and evaluating Pinch and i80 using feedback from peer software developers.
BIBLIOGRAPHY


