

University of Waterloo

Faculty of Engineering

Department of Electrical and Computer Engineering

Distributed Instant Messaging System

Final Report

Group #2015.010

Professor Werner Dietl (Consultant)

Qi Liu (20358515)

Asif Arman (20349964)

SangHoon Lee (20357600)

Danny Yan (20387735)

February 12, 2015

Abstract

Instant messaging systems of today simply cannot adequately safeguard the privacy of their users. Users' contacts lists, profile data, and even message histories are always logged and stored on centralized servers fully-controlled by the messaging system's service providers. There is simply no guarantee that users' data won't be searched and abused by these companies. Even if we put aside the trust issue between customer and service provider, a data breach by malicious third-parties is a dangerous, ever-present possibility on any centralized server exposed to the internet. Furthermore, recent leaks provided by Edward Snowden on NSA's overreach in its information collection practices has highlighted the fact that governments can easily and legally force service providers to hand over any and all user data available to them. Our system implements a truly decentralized peer-to-peer architecture with no centrally controlled servers of any kind. Unlike existing centralized messaging systems, messages in our distributed system will thus travel directly from the sender to the recipient, through a completely encrypted channel, giving no opportunity for any third-party to access message contents.

Acknowledgements

Foremost, we would like to express our sincere gratitude to our Assistant Professor, Werner Dietl for supporting our design project, and for being our consultant.

We would also like to extend our thanks to the authors and contributors to the open-source libraries we're using in our application, as well as the entire open-source community for making their work publically available for us to reference and build upon. We would like to give back to the community by releasing our source code for this project in an open source repository as soon as we feel it's ready for a proper release.

Lastly, we would also like to thank Professor Dan Davison for teaching the course ECE498a and ECE498b in which it lead to our creation of our project and our team.

Table of Contents

Abstract.....	1
Acknowledgements.....	2
List of Figures	4
List of Tables	5
1. High-Level Project Description of Project	6
1.1 Motivation.....	6
1.2 Project Objective.....	6
1.3 Block Diagram	6
2. Project Specifications.....	8
2.1 Functional Specifications	8
2.2 Non-Functional Specifications	9
3. Detailed Design	11
3.1 Graphical User Interface Design	11
3.2 Client-Side Services Design	12
3.2.1 Distributed Message Transmission Service Design.....	13
3.2.2 Distributed User Identity Service Design	17
3.2.3 Storage Service Design.....	19
3.2.4 Local Encryption Service Design.....	23
4. Prototype Data.....	29
4.1 Prototype Screenshots.....	29
4.2 User Interface Responsiveness Data.....	30
4.3 Continuous Integration Test Data.....	34
4.3.1 Peer To Peer Messaging.....	34
4.3.2 Peer to Multi-Peer Messaging	34
4.4 Resource Utilization Efficiency Data	35
5. Discussion and Conclusion	37
5.1 Evaluation of Final Design.....	37
5.2 Use of Advanced Knowledge	37
5.3 Creativity, Novelty, Elegance	38
5.4 Quality of Risk Assessment	38
5.5 Student Workload	40
References	42
Appendix A: Completed Prototype Hazard Disclosure Form.....	43
Appendix B: Completed Symposium Floor Plan Request Form	44

List of Figures

Figure 1: Block diagram of project components and inputs/outputs.....	7
Figure 2: How Tor Works [3]	14
Figure 3: Telehash vs Tor latency comparision	15
Figure 4: Dropbox vs remoteStorage.io upload performance	23
Figure 5: SJCL performance on various platforms	25
Figure 6: Crypto.js performance on various platforms.....	26
Figure 7: Prototype conversation screen.....	29
Figure 8: Prototype contacts panel.....	29
Figure 9: Prototype login screen.....	30
Figure 10: Prototype contacts status	30
Figure 11: Interface rendering performance measurements	31
Figure 12: Non-throttled loading performance of prototype	32
Figure 13: List of network throttling options for Chrome developer tools	33
Figure 14: Throttled loading performance of prototype	33
Figure 15: Average message delay for prototype	35

List of Tables

Table 1: List of functional specifications	8
Table 2: List of non-functional specifications	9
Table 3: Problem parameters for cryptography performance testing	24
Table 4: Average performance comparison between SJCL and Crypto.js	27
Table 6: Bandwidth usage measurements for prototype	36
Table 7: Table illustrating the number of hours invested for each student	40

1. High-Level Project Description of Project

1.1 Motivation

Instant messaging systems of today simply cannot adequately safeguard the privacy of their users.

Users' contacts lists, profile data, and even message histories are always logged and stored on centralized servers fully-controlled by the messaging system's service providers. There is simply no guarantee that users' data won't be searched and abused by these companies.

Even if we put aside the trust issue between customer and service provider, a data breach by malicious third-parties is a dangerous, ever-present possibility on any centralized server exposed to the internet.

Furthermore, recent leaks provided by Edward Snowden on NSA's overreach in its information collection practices has highlighted the fact that governments can easily and legally force service providers to hand over any and all user data available to them.

Some believe that the widely popular instant messaging network, Skype, is can protect the privacy of its users, as it makes use of a peer-to-peer architecture. However, this has not been true of the network since 2012, when Microsoft replaced all of the decentralized supernodes in the Skype network (peers that had enough resources to act as relays for traffic between other peers) with servers under their control [1]. A quick look at the current Skype privacy policy verifies that Microsoft indeed reserves the right to collect "Content of instant messaging communications, Voice messages, and video messages" [2].

As of today, the demand for truly private instant messaging systems has yet to be met.

1.2 Project Objective

We aim to design an instant messaging system that protects the privacy of its users as an utmost priority.

Our system will meet this objective by implementing a truly decentralized peer-to-peer architecture with no centrally controlled servers of any kind.

Messages in this system will thus travel directly from the sender to the recipient, through a completely encrypted channel, giving no opportunity for any third-party to access message contents.

User data is to be stored locally, or optionally on a cloud storage service for remote-access purposes, but always in a securely encrypted form that can only be accessed with the correct user-defined password.

1.3 Block Diagram

Our high-level application design, outlined in the block diagram in Figure 1, consists of a tiered architecture. The 3 tiers are the Graphical User Interface tier, the Client-Side Services tier, and the Execution Environment tier.

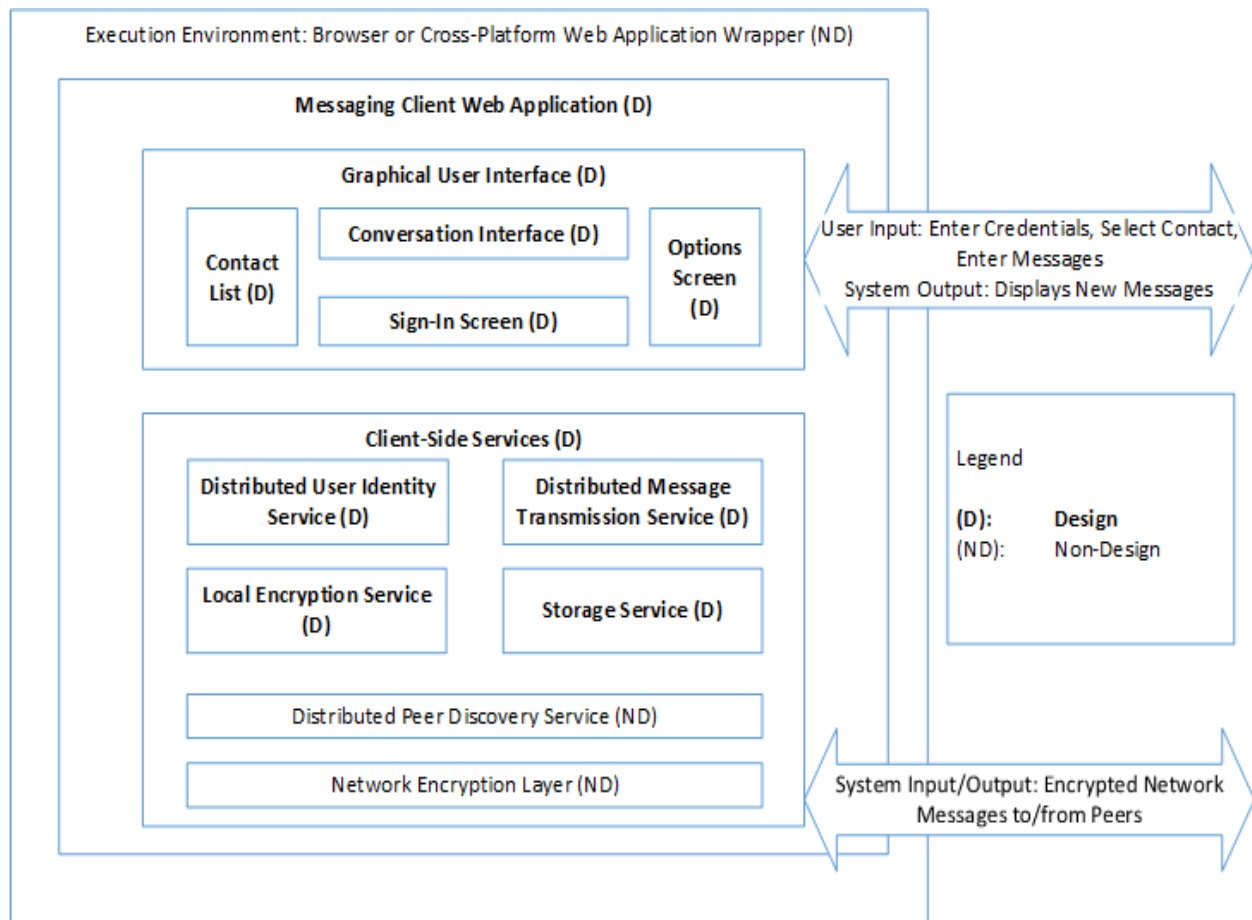


Figure 1: Block diagram of project components and inputs/outputs

The topmost tier consists of the Graphical User Interface (GUI) module and its various components. This tier is responsible for handling all user input such as logging in, selecting contacts, typing messages, and changing options, as well as displaying output from the system such as new messages received or contact availability status. The GUI tier also communicates to services in the Client-Side Services tier below in order for the system to execute actions based on user input, and display information based on system output.

The middle tier consists of the Client-Side Services module and the various services that it provides to the GUI tier above it. The Distributed User Identity Service maintains the contact list and profile information for the user, the Distributed Message Transmission service sends/receives messages during a conversation, the Local Encryption Service and Storage Service interact with storage mechanisms on the Execution Environment to persist user data between sessions in a secure manner.

The lowest tier consists of the Execution Environment module, which can be either a Web Browser, because our application is built as a Client-Side Web Application using only web technologies such as HTML and JavaScript, or a Cross-Platform Web Application Wrapper library that allows us to port the Web Application easily onto multiple Mobile platforms.

2. Project Specifications

2.1 Functional Specifications

Table 1: List of functional specifications

Functional Specification #	Essential / Non-Essential	Description
1	Essential	The user can send plain-text messages to other users and receive messages from other users in a reliable and robust manner, with no lost messages and less than 2 second delay under ideal network connectivity.
2	Essential	The application should not require the use of any centralized servers to store user data or send/receive messages to other peers.
3	Essential	Every message sent over the network should be encrypted. Messages should use at least 128 bit key for its encryption scheme.
4	Essential	Users must successfully authenticate themselves before they're able to access any private information such as the contact list or user profile.
5	Non-Essential	Contact list will have an accurate availability indicator for each contact. User can manually set his own availability status to any value when available.
6	Non-Essential	Users are able to participate in up to 10 conversations concurrently.
7	Non-Essential	Users can have access to the same contact list on every device that they authenticate into.
8	Non-Essential	The user can choose to send and display messages in real time. *See below.
9	Non-Essential	Offline message queuing will be supported. If the receiver of the message is offline (unavailable), messages can be queued, and the receiver will receive the message when he/she next becomes available.
10	Non-Essential	Group chat will be supported. Up to 4 users should be able to participate in the same conversation with no noticeable performance degradations.
11	Non-Essential	The user can choose to save conversation history. When a user reconnects to their account (on a different device) they will be able to view their previous conversations between the other users and groups.
12	Non-Essential	A user should be able to use the application on more than 1 device concurrently. State between each instance of the application should be synchronized to within 5 seconds of delay under ideal network conditions.
13	Non-Essential	Users should be able to search through their contact list and

		messages. Searching will be done in real time and filter through new messages as they arrive.
14	Non-Essential	Real time widgets will be supported that allow users to perform more than just sending messages. Examples may include a drawing widget, voice communication widget, video widget, etc.

The real-time messaging component at #8 in Table 1 was at one point an Essential specification for our application. However, we have investigated the performance characteristics of several distributed networking libraries and none of them could satisfy the performance constraints we deemed necessary for the real-time messaging experience to remain fluid and natural. In the end, we felt that real-time messaging was a relatively insignificant feature when compared to the disruptive privacy benefits that having a distributed architecture would bring. As such, we have changed the main focus of our application from real-time messaging to the distributed architecture itself. We have demoted the real time specification into a non-essential spec for the moment, but will continue to investigate performance characteristics of additional networking libraries to hopefully implement this feature in an acceptable manner in time for our final prototype.

2.2 Non-Functional Specifications

Table 2: List of non-functional specifications

Non-functional specification	Essential / Non-Essential	Description
Efficiency	Essential	Efficiency is one of the most important non-functional specifications that our system requires. The system needs to be as efficient as possible in both local and network resource usage. In terms of network usage, on average it should not surpass 10KB per 1000 characters sent/received. In terms of local resource usage, it should never use more than 50MB of memory under normal operation as a mobile or web application. Steps should be taken to minimize computational resource use on mobile systems whenever possible to conserve power.
Security	Essential	Our system needs to safeguard the privacy of the user as an utmost priority. All messaging traffic needs to be encrypted and resist tampering and eavesdropping. Only the intended recipient should be able to decrypt the messages efficiently with his private key, and be able to validate that it came from the expected sender. All local and storage need to be encrypted with a secret password provided by the user. Communication between clients must be accomplished without the use of a centralized server to store information or to route messages between them.
Dependability	Essential	Our system needs to be very dependable. Instant messaging systems

		are required to be very reliable and robust. Therefore, all messages should be delivered in the correct order and without any errors under ideal network conditions.
Usability	Essential	Our system should be highly usable, with an intuitive and responsive interface. Our system needs to have minimal input latency under all supported platforms. There should not be more than 0.5 second of lag between user input and interface response (not counting network latency) as long as the system has sufficient local resources for normal operation.
Maintainability	Non-Essential	Our system needs to be easily maintainable so we can update the client and add features without needing to perform architecture overhauls. Our component topology needs to be laid out in a layered architecture with high degrees of separation, with automated tests covering all important features to detect regression issues when changes are made.
Portability	Non-Essential	As our system may need to run on multiple platforms, portability becomes important aspect of our system. It should be able to run on android, iOS mobile platforms in addition to the original web platform, retaining all of functional and non-functional specifications. Once we have completed the core specifications for the web application, this can be accomplished using open source libraries that wrap the web application with platform specific interfaces.

3. Detailed Design

3.1 Graphical User Interface Design

This section of the document will describe the user interactive with our final product. It will describe how the application will display its content and the technologies that will be used to achieve them.

First, our messaging application will have sliding menus. Users can slide in and out the menus, swiping from left to right and right to left. The left menu will have a login form when the user is not logged in otherwise, a user profile display will be displayed. The purpose of the login form is to authenticate the user, which is from the essential specification #6. Also, this menu will have the contact list, a scrollable field that shows all the user's contacts. Regarding to the essential specification #5, each contact on the list will have an indicator of availability. There will be a unique color to indicate the status of that contact. In addition, these categories are implemented as tabs therefore, when selecting a tab, the content of the category will be expanded to display its content. For example, if the user is viewing his contacts list, the profile tab will be collapsed and the contact list tab will be expanded. This can maximize the space usage of the screen and make the application more dynamic and user friendly. Also, within the contact list tab we will have a "search" text input field on the top of the contact list, which will filter the user input in searching for contacts. For example, if the user is searching for the name, "Arthur", the moment the user types "a" it will automatically update the contacts list, showing the contacts that only starts with "a". This can make the search more advanced and will make the application dynamic and more users friendly. The search feature satisfies non essential specification #15. Moreover, there will be another row right below the contact search bar, which consists of two buttons. The first button will be used for sorting the contacts by name. While the second button will be used to sort the contacts by groups. The two buttons allow the user to specify his/her own preferences when searching for contacts. In addition, users will be allowed to group the contacts by right clicking on contacts and clicking on the add group option. Furthermore, invites button will be added in the last row as a static row. When user clicks on it will bring up a dialog with two options: invite by email and invite by ID. Invites by ID will only work if the ID exist so when the invites by ID is pressed we prompt the user for the ID and allow the user to send invitation only when the ID is found.

Moreover, on the right sliding menu, it will have options. There will be account setting option, storage switch option, saving chat history option and real-time messaging switch option. The account setting will allow users to change their nickname, password, and other personal information. The storage setting will allow the user to choose the storage options between cloud and local. Furthermore, the storage setting will satisfy the essential requirement #4 where the user is able to access the same contacts on every device that they sign in to. The saving chat history option allows the user to specific whether or not to save the chat history. This satisfies the non-essential specification #12. The real-time messaging switch will allow the users to choose between real time communication or not.

The sliding menu that will be implemented will use angular snap.js, which is the wrapper around snap.js that helps creating user-friendly shelves with handy styles and angular directives. By using this we can easily make the complex UI and handle the associated events in more organized manner. Furthermore, for the search mechanism, we will use regular expressions to match the strings and filter out the name

of the contacts. This will make the search feature more user interactive by filtering the contacts on every update.

In addition, the main screen, conversation interface, consists of two pages. One of the page displays all the recent chats with different contacts and the other page contains the chat history of the selected contact. Each of the contact on the most recent chat page contains the contact's name and the latest message delivered or sent. Inside the chat history page, the user is able to view the previous messages sent and received by the selected contact. In addition, there will be a text input field at the bottom of the chat screen for the user to communicate with the contact. To achieve the essential specification #3, we can add multiple tabs for different contacts. The multiple tabs will allow the user to participate in multiple conversations concurrently. In the chat screen, the user's messages will be colored in a different color from the contact's messages. The user's messages will be bounded to the right side of the chat screen with a unique color tag. While the contact's messages will be bounded to the left side of the chat screen with a different unique color tag. The color tags are used for the users to easily differentiate which messages are from them and which messages are from the contact. Also, each message will have a time stamp and be grouped by days.

Furthermore, styling of our application will be done using SaSS and compass with twitter bootstrap. SaSS is basically an extension to CSS that allow us to write CSS in organized manner. It will allow us to create variables so that we do not have to memorize or look up the values such as colors or sizes that are being used repeatedly. This can also improve maintainability because when we want to update or change certain colors or sizes we just need to change the value of that variable and it will propagate to the associated components. In addition, SaSS allows us to write advanced style sheet, create nested elements, which improves the readability, and build mixins, which are functions in CSS. By using mixins we do not need to write redundant codes. For example, there are multiple classes that require the styling attributes shown below.

```
-webkit-border-radius: 10px  
Border-radius:10px  
Background-clip:padding-box
```

Using SaSS, we can refactor the repeated styling attributes into a mixin and include this mixin wherever we want to apply round edges. This will save us lots of our time and can reduce the huge amount of redundant codes. Moreover, compass is basically a pre-built mixins that runs on top of SaSS and provide us the framework for SaSS. This will improve our workflow and speed up our development. Incorporating compass into our work will reduce unnecessary work since most of the mixins are already provided by compass.

Moreover, twitter bootstrap provides us with a framework that allows us to write very advanced and responsive styles. It will automatically handle the different screen sizes and it changes the layout of the screen with the respect to the screen size. Also, It has a lot of built in styles that looks professional.

3.2 Client-Side Services Design

Client-Side Services in our application will be implemented using the concept of a Service in AngularJS. AngularJS Services allows us to hide away implementation details of each Service module behind a

public interface, a concept in object-oriented programming known as abstraction. Other modules (even other Services) can call upon this interface simply by declaring the Service as a dependency. Below is a JavaScript code snippet showing the declaration of the Distributed User Identity Service (named “Identity”) that requires a dependency on the Distributed Message Transmission Service (named “Communication”) and Storage Service (named “Storage”):

```
angular.module('rtmsgApp')
  .service('Identity', function Identity($rootScope, Storage,
    Communication) {
    // Implementation of Service goes here
  });
```

A major practical benefit of using AngularJS Services, and abstraction in general, is that we can change the implementation details of some functionality in a Service without having to change the interface to that Service, so any existing code that calls on the interface will continue working as if nothing has changed at all. This enables better separation of concerns between application modules and allows for better Maintainability of the application as we will inevitably need to make changes to our Services throughout the development process (satisfies Non-Functional Specification “Maintainability” in Table 2). The Storage Service detailed below is a more concrete example of the benefits of abstraction, as it switches out the entire implementation of its interface methods between a Local Storage service and a Cloud Storage service depending on user configuration.

3.2.1 Distributed Message Transmission Service Design

The Distributed Message Transmission Service, henceforth to be referred to as Communication Service, is responsible for the core operation of our application, delivering messages from one user to another in a distributed, peer-to-peer manner without involving any centralized servers.

We have investigated two major designs for the Communication Service, one based on the Tor Anonymity Network and another based on the Telehash Distributed Hash Table (DHT) implementation.

3.2.1.1 Tor Anonymity Network

From the description on their website, the Tor Anonymity Network is a “free and open network that helps you defend against traffic analysis, a form of network surveillance that threatens personal freedom and privacy, confidential business activities and relationships, and state security”. [3]

Tor attempts to achieve total anonymity of its users by encoding all source and destination internet addresses into a Tor node address ending in .onion, and routing traffic through a series of randomly selected Tor relay servers run by volunteers across the globe. All packets are encrypted with a 256-bit key in multiple layers (hence the onion on their logo), and at each hop in the series, only a single layer is decrypted as to obtain the destination address of the next hop the packet is to be routed to, and nothing else. This way, no single relay on the path from source to destination can have full information on the original source or intended destination of any packet travelling through it. Packets sent through the Tor network do not need to adhere to any specific format because it allows for the transmission of unaltered TCP/IP packets, but the extra obfuscation and encryption/decryption layers do add a non-negligible overhead to packet size and delivery speed. [3]

Figure 2 below illustrates how this process may look like for a given path through the Tor network between two clients:

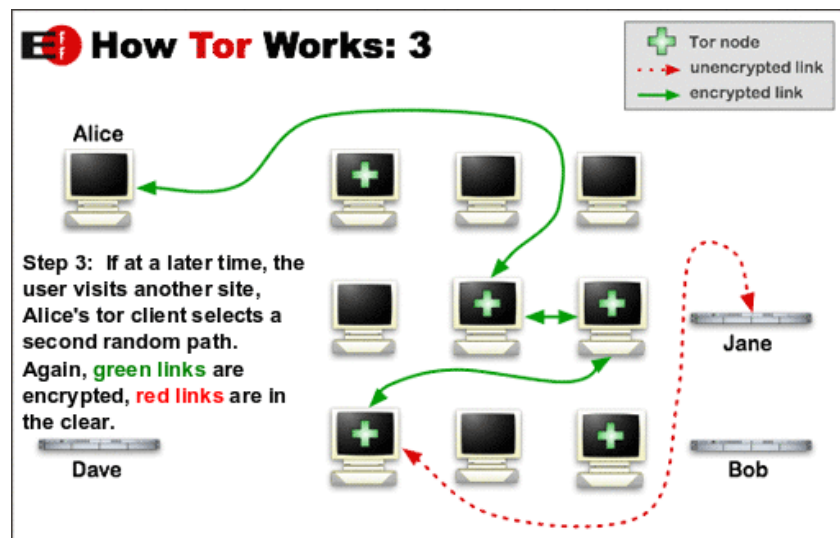


Figure 2: How Tor Works [3]

3.2.1.2 Telehash DHT

Telehash claims to be “a secure wire protocol powering a decentralized overlay network for apps and devices”. [4]

Telehash’s decentralized nature is based on of a concept in distributed computing known as Distributed Hash Tables (DHTs). Similarly to a regular hash table in computer science, the DHT also provides an efficient lookup service for key-value pairs, but unlike a traditional hash table, DHTs store their key-value pairs across multiple nodes in a distributed manner, and each node is responsible for maintaining a certain portion of the key-value pairs in the table. When initiating a lookup, a node will query the DHT network for the peer responsible for the key it’s searching for. [4]

Telehash provides on top of a regular DHT, an end-to-end network encryption framework using length 256 bit keys, a uniform message format, and an automatic peer-discovery and packet routing system. Each user is provided with a public/secret keypair on initialization, and an id is then generated based on this key pair that is used to identify and communicate to the user on the Telehash DHT network. Packets sent through Telehash must adhere to the JavaScript Object Notation (JSON) format, which encodes messages in a string format similar to the syntax of JavaScript object declarations. [4]

3.2.1.3 Solutions Analysis

At first glance, both Telehash and Tor are able to meet functional specifications 1, 2 and 3 in Table 1, and the Security non-functional specification in Table 2. They are both completely decentralized communication solutions that are able to carry messaging traffic as payloads from one peer to another, securely encrypted with a 256 bit key without the use of any centralized servers. However they do have some very key differences in their performance characteristics and implementation details that ultimately led us to choose Telehash as the base library to implement our Communication Service.

To test the performance of Tor compared to Telehash we set up 2 simple projects that sends and receives a message of fixed length. The first one uses the Telehash Javascript library while the other project uses node-Tor, which is a Javascript implementation of Tor. The test was set to measure the latency to send 1 message (not including the initialization time it takes to connect to the Telehash/Tor network). The latency was recorded over several test runs, the results can be found in the following graph:

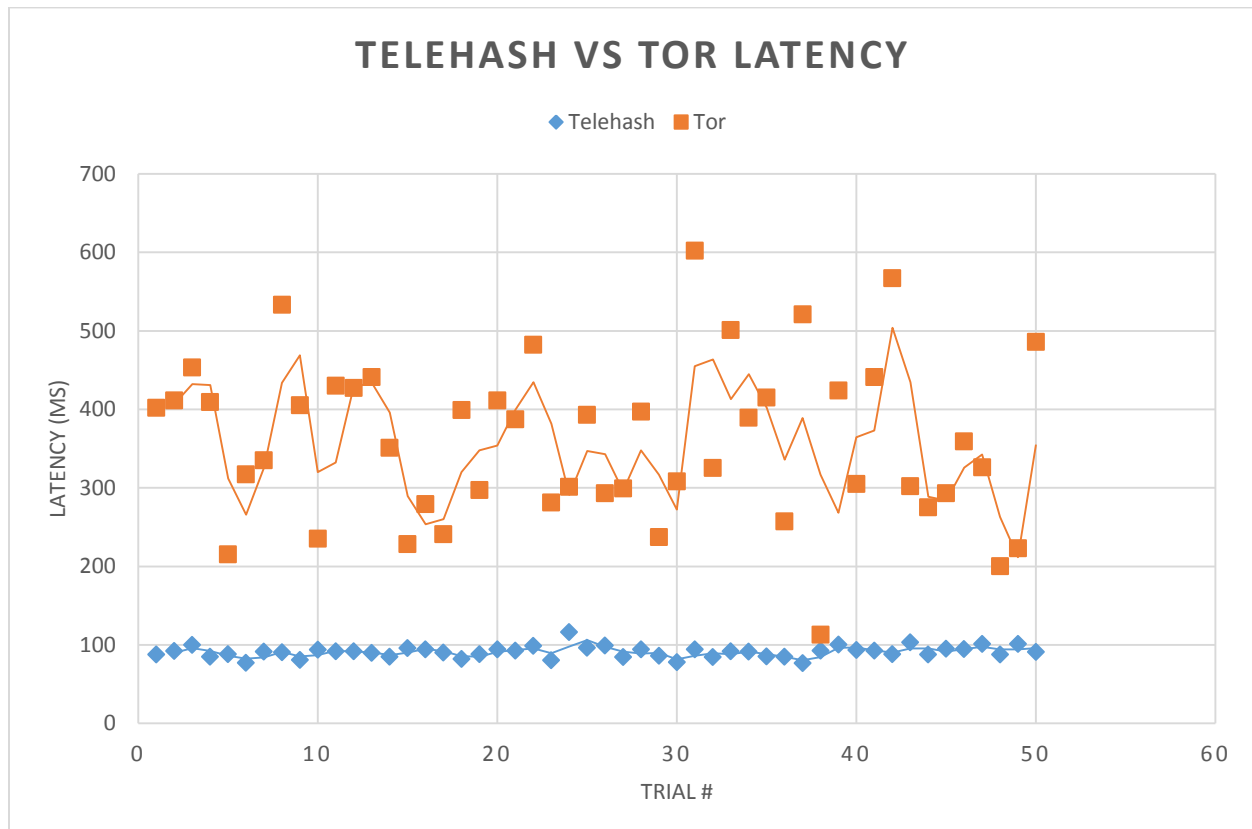


Figure 3: Telehash vs Tor latency comparison

From the results, we have verified that Telehash generally adds less than 100ms latency on average on top of the base round-trip latency (obtained through the ping command in windows command line) for any given traffic, whereas Tor's latency overhead tends to be much higher and much more unpredictable, ranging from 200ms to 500ms. This can likely be attributed to the higher degree of origin and destination obfuscation that the Tor network performs with its multi-layered encryption setup and random routing path. No packet loss was detected for the duration of the test where network connectivity was uninterrupted, so both solutions adequately satisfy the Dependability non-functional specification in Table 2.

For the purpose of our project, we aim to provide the maximum possible privacy protection to our users, so Tor's extra protections against communication metadata monitoring is definitely desirable despite its high latency overhead. However, after some additional research, we discovered that there was an extension available for Telehash called Opaque Routing Tree (ORT) that enables similar routing

obfuscation and layered encryption as Tor [4]. The latency characteristics of Telehash with the ORT extension enabled was discovered to be similarly unpredictable, and slightly worse than Tor, ranging from 300ms to 700ms. However, we believe the extra flexibility in being able to enable/disable the feature as an extension to achieve higher performance on demand is definitely a valuable feature to have as well for possible timing-sensitive extensions to our application.

In summary, the tradeoff we have to make here is between Tor's marginally better absolute performance when routing obfuscation is enabled (always, in Tor's case) versus the higher degree of flexibility that Telehash provides with its optional ORT extension. We believe the latter is more valuable to our application because it the higher performance characteristics when ORT is disabled makes real-time communication much more feasible for our system, paving the way for the successful implementation of functional specs 8, 12, and 14 in Table 1, and also potentially improving the Usability non-functional specification in Table 2.

Another aspect of our testing involved packet size overheads in addition to latency. As mentioned before, Telehash enforces the use of JSON for messages transmitted over its network, while Tor has no similar restrictions, and thus can be used with more compact binary message formats. We tested the size overhead of the Telehash JSON packets over a custom encoded binary message holding the same data (approximately of approximate length 100 characters). The overhead proved to be over 100%, with the JSON packet weighing 309 bytes while the binary message was only 138 bytes.

Clearly the binary encoding allowed by Tor is superior in terms of the Efficiency non-functional specification in Table 2. However, binary encoding ends up being an extra computational step over JSON encoding because all data handled by our application will be plain JavaScript objects, that naturally adhere to the JSON format, this offsets the advantage of Tor in terms of Efficiency as power usage is another important aspect in efficiency to consider, especially for mobile platforms, where power concerns may outweigh even network usage. It is also important to note that 309 bytes for 100 characters is still well within our specified upper bound of 10KB per 1000 characters for the Efficiency specification.

The final factor that went into our decision to choose Telehash over Tor is the Portability and Maintainability non-functional specs in Table 2. Ideally, our application should be a completely self-contained web app with no dependencies on any software that need to be installed on the local system, other than perhaps a web browser. This allows us to freely port the application from the web to any mobile platform through the use of a wrapper library like Adobe PhoneGap [5]. Telehash makes this possible because it has multiple library implementations, including one in JavaScript, so we can simply reference the library in our project to make use of it. However, Tor makes this impossible as the only implementations available are based on network proxies that must be installed on the local system to route traffic though the network. Although these proxies are available on multiple platforms including mobile, we believe the requirement to install a separate piece of system level software will significantly deter the adoption of our software by the general public.

3.2.1.4 Final Design

We plan on implementing the following interfaces to the Communication Service using Telehash:

```
Communication.initialize()
```

This interface will be a low level method used to initialize a new user on the Telehash network, complete with a new encryption Keypair and user id. It is meant to be used by the Identity Service (see below) in initializing a new local user profile, which will then be populated with other information such as the user's display name and local encryption password.

```
Communication.connect(user)
```

This interface will be used to connect a given user to the Telehash network, similar to the Sign In paradigm of existing messaging systems. The user object is a JavaScript object of the user type, the structure of which is detailed in the Identity Service below. Once connected, a user will be able to send and receive messages using the Telehash network.

```
Communication.send(contact, message)
```

This interface will be used to send messages to another user on the Telehash network. The contact object is a JavaScript of the contact type. The contact object will contain among other things, the user id for that particular contact, which Telehash will use to identify peers and route messages. The message object is a JSON object containing the contents of the message. In the implementation of this service, we will use Telehash to encrypt the message and send it to the intended contact over the network.

```
Communication.receivedMessage(contact, message)
```

This interface is an event handler for handling messages received by other peers on the Telehash network. It will be called whenever a message is received, and will trigger an event in our app to be handled by the GUI layer to display a new message to the user. The user object will contain the Telehash id of the user that sent the message, and the display name of the current user. This id will be passed to the Identity service to perform a lookup against the user's contact list to fetch additional information for the contact, such as the display name. The message object is a JSON object containing contents of the message received.

3.2.2 Distributed User Identity Service Design

The Distributed User Identity Service, henceforth to be referred to as the Identity Service, is responsible for keeping track of information about the current user, as well as managing the user's list of contacts. It will require a dependency on both the Storage Service for accessing and storing persistent data as well as the Communication Service for creating a new user on the Telehash network.

We will implement following internal data structures for the Identity Service:

```
Identity.currentUser = {  
  id: '...', //telehash user id for current user  
  name: '...', //user's chosen display name  
  password: '...', //user's chosen local encryption password  
  keypair: '...' //telehash secret/public keypair used for  
  network traffic encryption  
}
```

The `currentUser` data structure is an object of the user type, containing the 4 properties shown above. This object is used to store profile information regarding the current user of the application. This is passed into the communication service when calling `Communication.connect()`.

```
Identity.contacts = {
  id1 : { // telehash user id for contact1
    id: '...', //telehash user id for contact1
    name: '...' //display name for contact1
  },
  id2 : { // telehash user id for contact2
    id: '...', //telehash user id for contact2
    name: '...' //display name for contact2
  },
  ...
}
```

The `contacts` data structure is a JavaScript object with properties containing the id of the contacts and with values of the contact type, with the two properties shown above. This object stores a list id-contact pairs for the user that is used to populate the Contact List GUI module, and allow our application to efficiently map ids fetched from the storage service to the actual contact objects that the application can process, within a constant time regardless of the length of the contact list, satisfying the Efficiency non-functional spec in Table 2.

Any contact in the list can be accessed in the following extremely computationally efficient manner:

```
Identity.contacts['id1'];
```

Whereby `id1` is a Telehash user id fetched directly from the Storage Service. More details regarding the implementation of the Storage infrastructure is available below.

In addition to these internal data structures, we will expose the following interface methods for the Identity Service:

```
Identity.updateUser = new function(user) {
  this.currentUser = user;
  return Storage.save('user', user);
}
```

This method will be used to update the current user stored in the service with the new user object passed in, as well as save the user to the persistent Storage Service under the 'user' key.

```
Identity.createUser = function(user) {
  return Communication.initialize().then(function (newUser)
  {
    this.updateUser({
      id: newUser.hashname,
      keypair: newUser.id,
      name: user.name
    })
  })
}
```

```

    });
    }.bind(this));

};

```

This method will be used to generate a new Telehash user id and keypair using the `Communication.initialize()` service method and then create a new user object using the user profile information from the user object passed into the method. It then persists this object in the Identity Service and Storage service using the previously discussed `Identity.updateUser` method.

```
Identity.getContact(contact)
```

This method will be used to fetch an existing contact in the user's contact list by the contact's Telehash id.

```
Identity.inviteContact(contact)
```

This method will send an invite to the Telehash Id of the contact object, along with a specially formatted JSON message, to be handled by `Identity.receiveInvite(contact)` shown below. It will also add a temporary contact to the contact list that will be promoted to a full contact once it receives a similar specially formatted JSON message (acting as acknowledgement) from the invited contact.

```
Identity.receiveInvite(contact)
```

This method is an event handler for the special JSON message sent from the `Identity.inviteContact()` method of another client. When the special JSON message is detected, the `Communication.receiveMessage()` method will pass the message to this event handler instead of triggering the message received event to notify the GUI layer. This method will then trigger an invite received event to notify the GUI layer of an invite from a new contact, and allow the user to accept or decline. If accepted, this method will then proceed to call `Identity.inviteContact(contact)`, passing in the sender of the invite as the parameter. This will complete the handshake processes and establish the two clients as contacts for each other in their respective instances of the application.

3.2.3 Storage Service Design

The storage component will be used to store the user profile for an individual user. This will handle the essential feature that allows user to access and save their contact list. The Storage component will be implemented as an Angular Module and will depend on 2 other Angular Module. These 2 services are the `LocalStorage` service and the `CloudStorage` service. The user will be able to configure whether to use `LocalStorage` or `CloudStorage`. If the user chooses `LocalStorage`, the users profile data will be saved locally on the browser, if the user chooses `CloudStorage`, the user profile will be saved server sided which allows the use of this application on multiple devices. The Storage service will support the following 4 functions as a minimum:

```

Storage.save(key, value);
Storage.read(key, value);
Storage.update(key, value);
Storage.delete(key, value);

```

This means the Storage service provides the application with key value storage. When one of these function is called, the Storage service will check to see the user configuration, if the user configuration is set to local storage, it will use the LocalStorage Service to save the data onto the local browser. If Cloud Storage is set, the CloudStorage service will be used instead to save the data server side.

To make the Storage service more flexible and easier to use, tables may also be implemented. This will make it easier when we want to save more than just contact list, such as saving messages sent and received (a non-essential feature). So there would be a Contact table, and a Messages table. The function would be:

```
Storage.save(table, key, value)
```

Also the Storage service will use the Local Encryption Service to encrypt the data before inserting it into the Local/Cloud storage. The detail of this is can be found in the Local Encryption Service section.

3.2.3.1 Local Storage Service

The local storage service is used to save the data onto the local browser. To save data locally on the users disk, HTML5 Web Storage will be used. These are similar to cookies but unlike cookies which are meant to be read on the server-side, web storage is meant to be read only on client side. Unlike cookies they do not take part in HTTP requests and have a much greater limit of 5 MB. The HTML5 Web Storage is currently available on all major web browsers.

In terms of the 5 MB size limit, if each Contact takes no more than 50 Characters as their user name, and their unique telehash Hashname is 64 characters, then each Contact requires 114 B of space. This means almost 9000 contacts can be saved if we only use 1 MB of space. There is clearly more than enough space if only contact user name and connectivity information is saved. If other data is to be saved such as message history, we can limit the size of the Contact list to 150 contacts, and use the rest of the space to store other information. For storing message history we will only keep track of the most recent messages and if the storage space is full, the new oldest message will be replaced with the newest message. Note that the final data that is actually stored will be encrypted meaning that size will actually be greater than what is specified here. We performed a variety of quick test with the encryption service by encrypting different files with different sizes and found that the size does not increase more than 30% which makes the storage system still viable.

LocalStorage is based on key value pairs, so it is simple to build this under the Storage service. The code to set a key item using HTML5 Web Storage is simple:

```
localStorage.setItem(key, value);
```

So When Storage.save(key, value) is called, if LocalStorage service is enabled, the above line of code will be used to set the Storage service. To create the illusion of tables using web storage, the table will be saved as the key, and the key value pair will be saved as an object under the value as shown below:

```
localStorage.setItem( table, { key : value } )
```

3.2.3.2 Cloud Storage Service

The use of local storage is limited to a single browser on a single device. To allow users to save their profile information along with the ability to synchronize profile information across devices, this data must be stored online. An alternative that was considered is to store user profile information into a database such as MongoDB or MySQL in our own hosted server. Due to large overhead and costs associated with this solution, we would like to avoid it.

The solution we ended up committing to, is to use a free cloud storage services that offers space on a per user basis. These services include Dropbox, Google Drive, remoteStorage, and SkyDrive. Each of these services have API's that provide functionality to allow users to save information by logging into these individual accounts. The main API's which is being considered are Dropbox's Datastore API [6] , Google Drive SDK [7], remoteStorage.io [8].

The Dropbox API has been investigated and tested. The DataStore API provided by Dropbox, allows the user to store records into multiple tables. The actual tables are schema less and records inserted are JSON style objects, meaning that the data stored is similar to the key value storage used by local storage. Dropbox currently has a limit of 10 MiB for its Datastore, which should be adequate since it is twice the size provided by local web storage.

Implementing this the Storage services using Datastore would require a default table, since Datastore uses tables by default. When `Storage.save(key, value)` is called, Datastore will save the key value pair in the 'default' table as shown below:

```
var defaultTable = datastore.getTable('default');
defaultTable.insert({
  key: value
});
```

When storing data for a specific table, the `.getTable()` function will just use the name of the table as shown above.

Google Drive SDK was also investigated, as they have a rich SDK that allows per user data storage. Drive SDK is more focused on uploading, modifying and reading files as a way of storing user data. When storing application data, Drive has a special hidden Application Folder in which we can store and read from files [7].

Upon comparing Drive SDK to Dropbox API, Dropbox was initially chosen as the preferred cloud storage solution. Our application is more suited to Dropbox's Datastore model which is more closely related to local storage making it simpler and quicker solution to implement. Google Drive SDK, currently has a limit of 10,000,000 queries/day which is a per app quota. Although Dropbox doesn't specify the exact number, the rate limit is done per user as opposed to per app, *"The API limits the amount of calls your app can make per user. The limits are high enough that any reasonable use of the APIs shouldn't come close to hitting them"* [6]. Therefore, between the two candidates of Dropbox and Google Drive, Dropbox will be selected as the cloud storage service.

However, unlike Dropbox or Google Drive, remoteStorage is uniquely different from those cloud storage service. Dropbox and Google Drive are centrally controlled, in which your data will owned and stored on their server specifically. Whereas remoteStorage is just simple open protocol which any developer to host it and implement on their own server. Therefore, the client's data are stored in the developer's server instead of a third party server. In addition, remoteStorage gives the ability of transferring data between different service providers and/or their own servers in a distributed manner. What remoteStorage can provide is much greater than Dropbox or Google Drive. Therefore, it was decided that remoteStorage was the preferred cloud storage service for this project.

```
remoteStorage.access.claim(scope, mode);
RemoteStorage.defineModule(scope,
    function builder(client) {
        return {
            exports: {
                client.declareType(
                    // necessary properties
                )
            }
        }
    }
);
```

To operate with remoteStorage, the user is required to claim access to a given scope in order to access or to create its data in the storage entity. After successfully claiming a successful scope, the user is able to define modules inside that scope in the data storage entity.

Adding the cloud storage component introduces a form of centralization that may seem to counter the distributed nature of our application. Although these user profiles are stored and synched from centralized servers, the nature of communication is still decentralized. Essentially if the user chooses to use cloud storage, they will retrieve their contact list information from the centralized servers. The messages sent using the contact list information will remain as direct P2P communication. Encryption will also be performed, user profile information will be encrypted before saving onto the cloud, and will be decrypted after retrieving back from the cloud. This means even if someone had access to the users cloud storage account, the data for the application would be useless to them without the correct password.

We have also performed a performance test on the two viable online storage candidates, Dropbox and remoteStorage.io, for the cloud storage service. The performance test is targeted on the latency of uploading data to cloud storage servers.

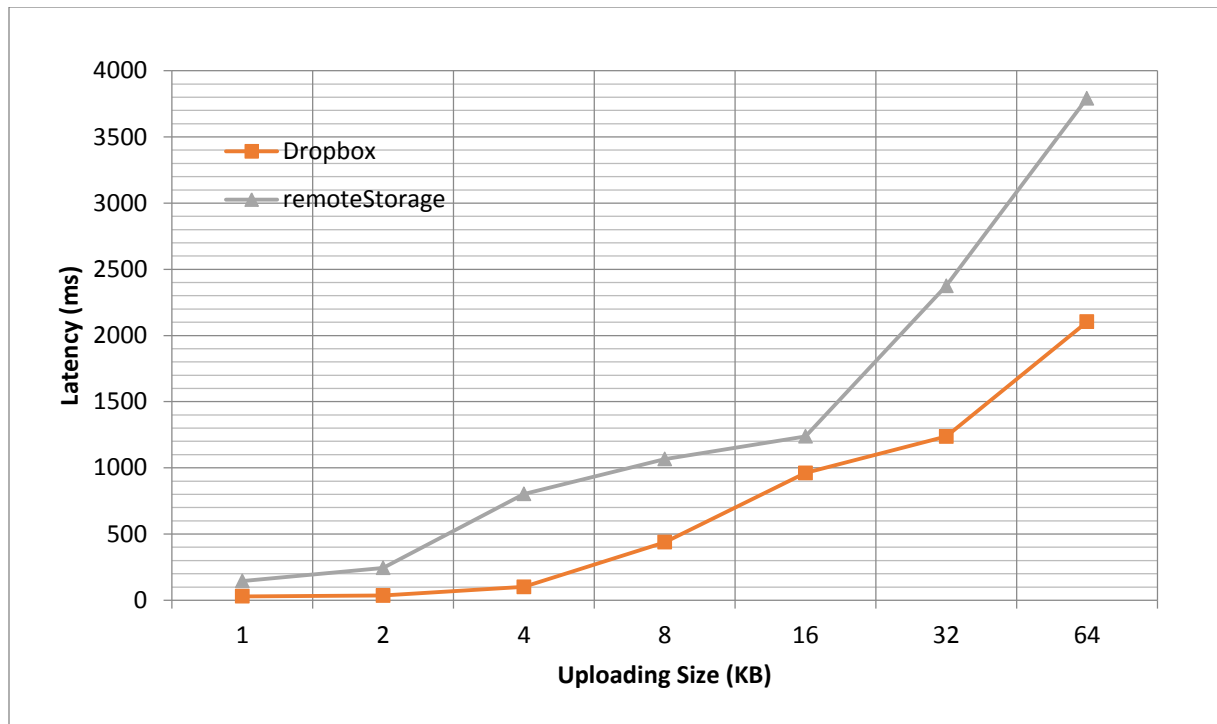


Figure 4: Dropbox vs remoteStorage.io upload performance

As shown in the figure above, it illustrates the latency required to transfer a certain data size. From the observation, Dropbox solution shows an average of 39% shorter latency than the remoteStorage solution. The result can conclude that Dropbox's performance in latency for uploading data is greater than the remoteStorage.

However, it is still arguable that the performance difference is not significant enough to select Dropbox as the solution for the cloud storage service. Due to the flexibility power of the remoteStorage where it has the ability to migrate the hosting server, and prevent the data being stored on a third party server promotes the project's main interest in protecting user's privacy. Another factor that does not make Dropbox the selected candidate for the cloud storage service is that the cloud service storage can be not used at all for the user. The main purpose of having a cloud service storage is to synchronize account information between different devices, not all user will want this. There will be a percentage of users who will not be depending on a cloud storage service. Therefore, the performance property of the solution for the cloud storage service should not outweigh the advantages of using remoteStorage.

3.2.4 Local Encryption Service Design

Security is a major component that is considered for this application. The storage service is used to store data into either a local storage on the browser or server side using a cloud based service. In using either of these services, data should not be stored in plain text. For local storage, it is less of a necessity, however if someone has access to the device, they can easily copy the contents of the local storage, so the data in local storage should be encrypted. For cloud based storage service, the only way to guarantee that the cloud service provider does not view the data is to encrypt it before saving it onto their server.

We have evaluated two crypto libraries for our application: Stanford JavaScript Crypto Library (SJCL) and Crypto.js. Since our application will be doing a significant amount of cryptographic operations, the performance of the crypto library will be a paramount to the performance of our application. This and the fact that the two crypto libraries are roughly equal in terms of features resulted in us focusing our testing on the performance aspects of the libraries.

3.2.4.1 Encryption libraries performance testing

The performance of the SJCL and Crypto.js crypto libraries was tested varying 3 different factors: number of rounds, key size, and authentication tag size. The increase in rounds involves more confusion and diffusion therefore it increases the security against cryptanalysis. The Increase in key size and tag size increases the security by increasing the possible combinations (ex. 128 key size: 3.4×10^{38} combinations, 256 key size: 1.1×10^{77} combinations). The following table shows the performance of encryption and then decryption of varying parameters for SJCL and Crypto.js.

Table 3: Problem parameters for cryptography performance testing

Test	
Default	# rounds: 20 Key size: 128 Tag size: 64
Medium	# rounds: 1000 Key size: 128 Tag size: 64
Hard	# rounds: 1000 Key size: 256 Tag size: 128
Super-hard	# rounds: 2000 Key size: 256 Tag size: 128

From the result, it seems like the trade-off of getting higher level of security is performance. The higher the strength of the security the lower the performance. However, the lowest level of cryptography that was tested, the default, was still secure enough that it was not worth to pay the performance for the security increase. In fact this is used in many real world application and yet have any security problems.

For portability and maintainability issues, the performance of the SJCL and Crypto.js libraries was tested on variety of platforms. Mostly on different versions of web browsers. The performance of each library is shown in detail on the following figures:

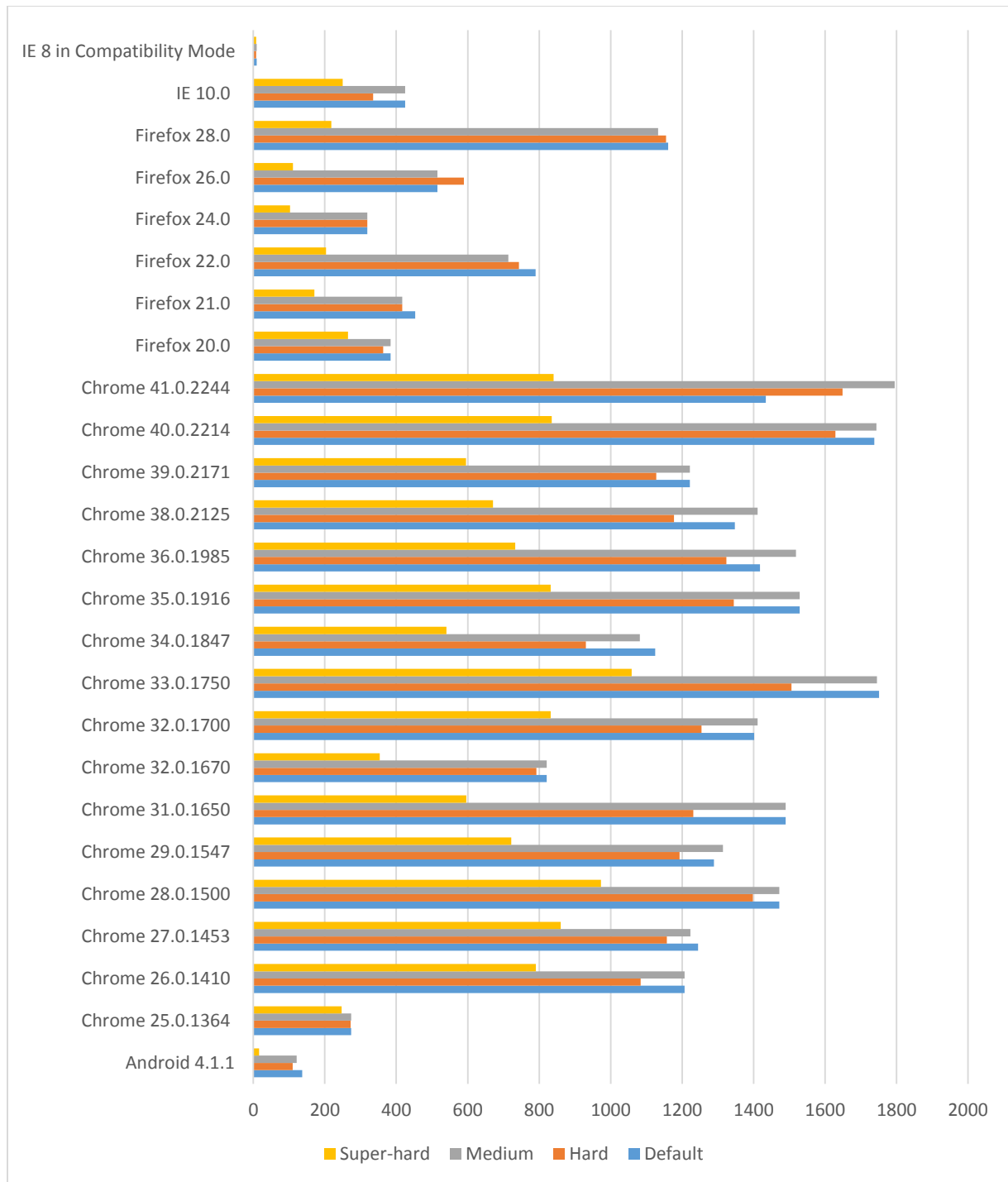


Figure 5: SJCL performance on various platforms

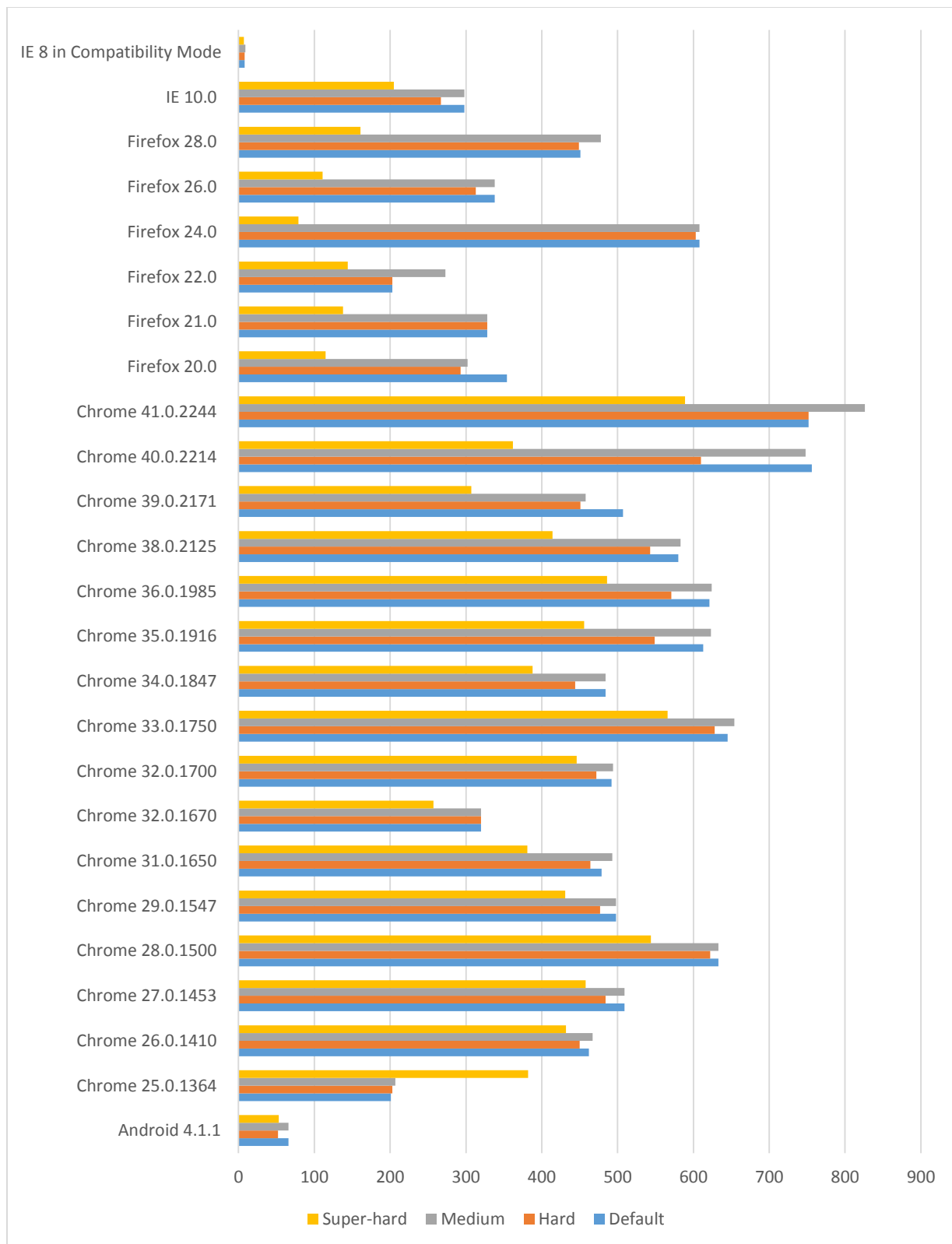


Figure 6: Crypto.js performance on various platforms

From looking at the Figures, Chrome browser performs the fastest and then the Firefox. Internet explorer less than version 10.0 and mobile platform seems to have very low performance. The general trend is that the version of the browsers and the performance are directly proportional. The average performance for all results can be summarized in the following table, with an increasingly higher weight placed on more recent browsers:

Table 4: Average performance comparison between SJCL and Crypto.js

Test	Operations/seconds
SJCL	637
Crypto.js	385

From looking at the testing result, SJCL performs approximately 2 times faster than Crypto.js.

3.2.4.2 Final Design

Due to its superior performance, we will use the Stanford Javascript Crypto Library (SJCL) [8] to perform local encryption on our app. The local encryption service will be built under this library. This library provides the necessary functions to encrypt and decrypt the data. The default encryption scheme provided in SJCL will be used, which encrypts using AES block cipher with a key size of 128 bits; this meets our minimum specification. SJCL also adds additional security parameters such as generating a random salt based on the password and hashing over multiple iterations (1000 iterations by default).

For the encryption system to work, the user is required to have a password. When a user first sets up his/her account, the user will be requested to set his or her password. Once this password is set up, the encryption service will be used to store it. The password itself will not be stored as a variable in the Encryption service module. Instead its hash value will be stored which can be done with SJCL since it supports SHA256 hashing. Once the hashed password is stored in the Encryption service, decryption and encryption of the data can be performed. As mentioned before, the Contact list is stored within the identity service as a JavaScript Object. The contact will be stored in the Storage service in a 'Contact' table. Each key value pair in the table will refer to an individual contact. The key for the contact will be the User ID and the value will be any other information related to that specific list. Currently this value is only the User Name, but other information may also be stored such as a display picture url and whether the individual belongs in a specific group. Before this data is saved into the storage, the Encryption Service will encrypt both the key and value pair using the hashed password.

This above mechanism makes it easy to create, update, add and delete Contact list information. For example if we want to delete a specific user, we will locate it using the encrypted User ID, and delete that key value pair from the LocalStorage / CloudStorage. A problem with this implementation is the fact that everything is based on a password which is not stored server side like traditional chat system. If the user forgets his/her password, there is no service to reset the password meaning the decryption of the contact list will not work which essentially makes the app unusable. This problem can't be resolved unless we provide our own service to store the user password which imposes cost and creates a centralized component that we would like to avoid. The best we can do is to place a warning to the user that the password can't be reset if forgotten. Another problem is if the user would change their password. This means that all the encrypted data that is stored must be read, decrypted and then re-

encrypted and saved back into Storage. This may be a time consuming process, but we assume users will not change their password frequently compared to other operations.

4. Prototype Data

4.1 Prototype Screenshots

Here are some screenshots of our prototype in action:

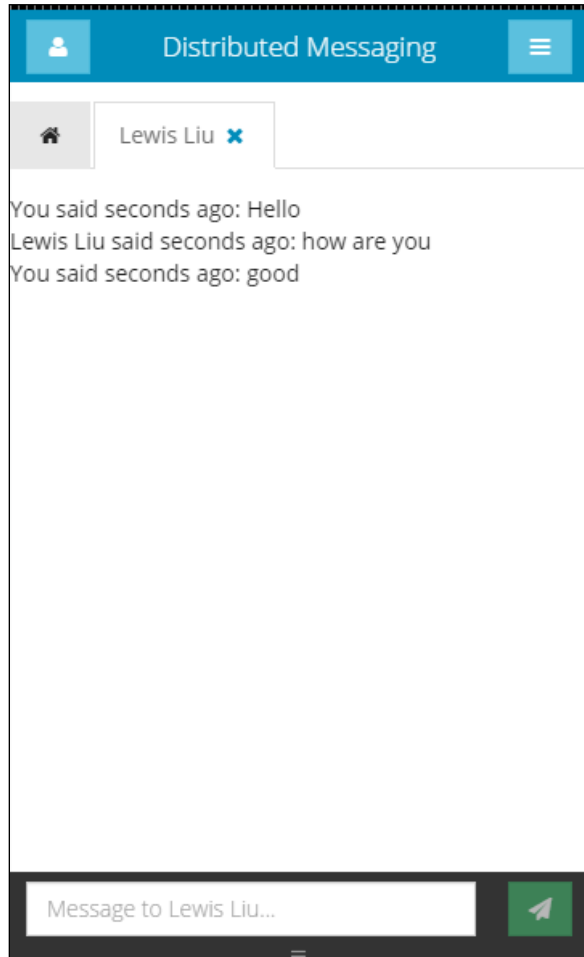


Figure 7: Prototype conversation screen

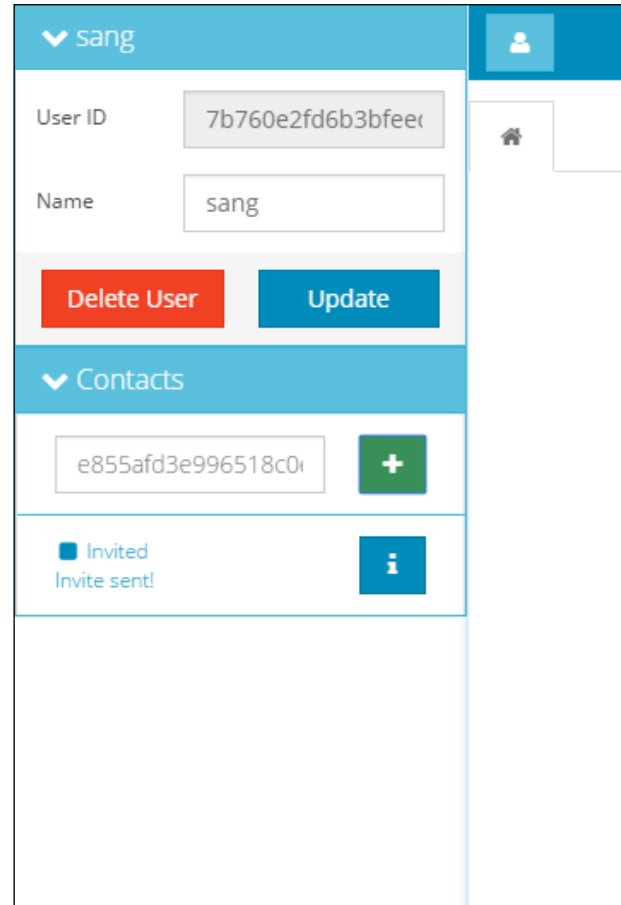


Figure 8: Prototype contacts panel

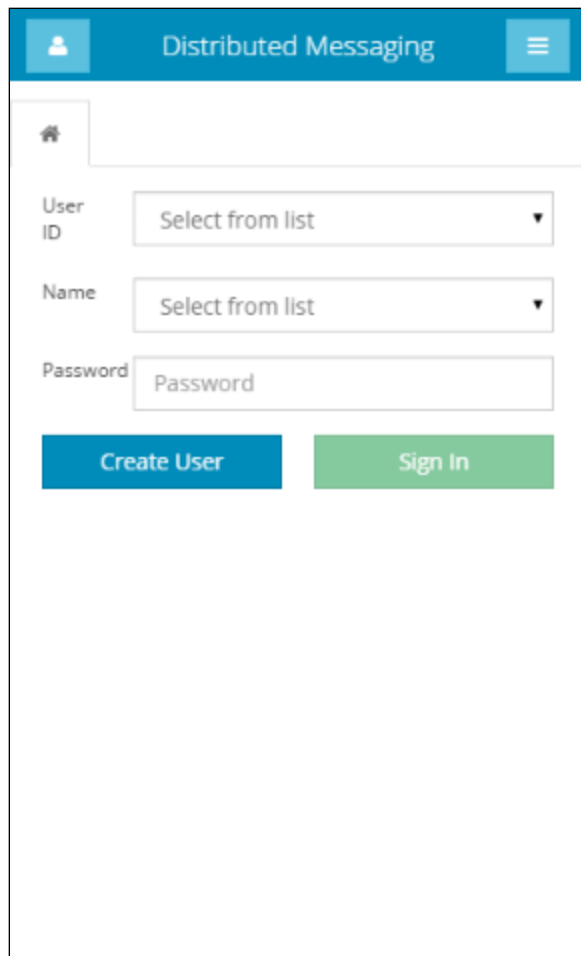


Figure 9: Prototype login screen

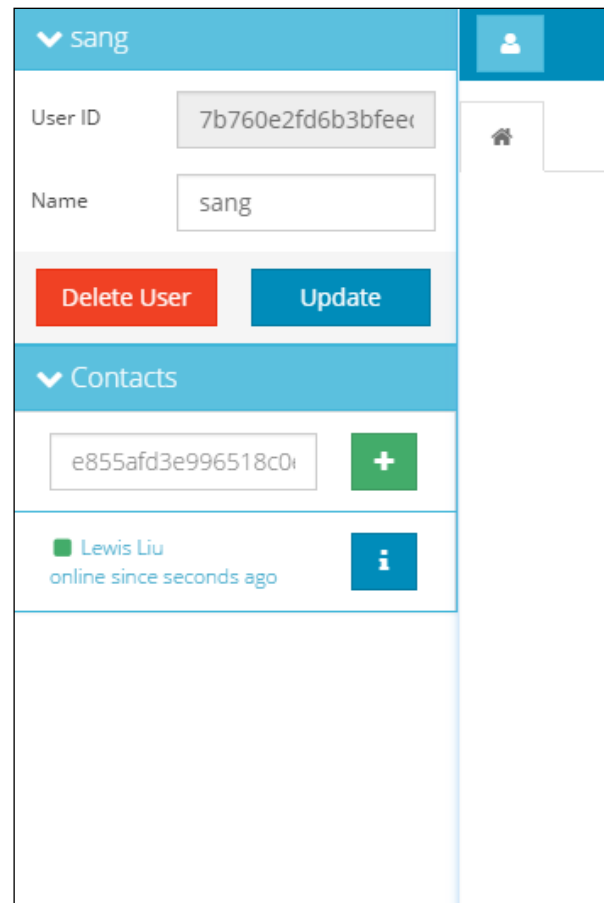


Figure 10: Prototype contacts status

4.2 User Interface Responsiveness Data

We realized that it was not enough that our interface only “feels” fast in use, so we’ve developed several methods of testing the responsiveness of our application in terms of quantitative metrics. We made use of the excellent Google Chrome development tools to obtain most of these metrics.

First of which is the interface rendering speed. We use Chrome’s rendering metrics tool to display an overlay on top of our app showing the number of Frames Per Second (FPS) the interface can be continuously redrawn. We then perform a specific array of user interactions on the app that result in dynamic UI response, and note the measurement of the FPS meter...

The following figure shows our interface rendering performance:

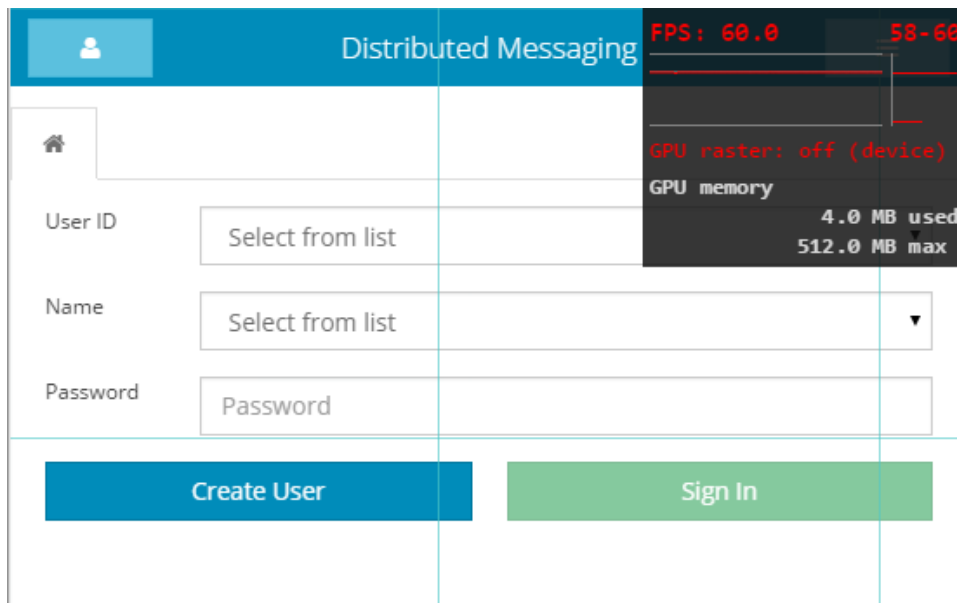


Figure 11: Interface rendering performance measurements

Every action we performed resulted in a very smooth 60FPS. 60FPS is the ideal frame rate to render applications on most screens since most LCD monitors refresh at rate of 60 Hz. Lower FPS than 60 means animations will look choppy due to the mis-synchronization between display refresh and application rendering, while rendering at more than 60FPS will simply be a waste of computing power because most LCD screens will not be able to render fast enough to display the extra frames anyways.

Another metric we're measuring is the initial loading time of our application. Our application consists of several complex, heavy weight third party libraries (such as telehash, which by itself weighs 700 kilobytes), as well as a large amount of custom logic, markup and styles totaling approximately 1.5 megabytes. To counteract this, we're using the free tier of the Cloudflare Content Delivery Network service to optimize the loading speed of all of our static page assets (including html, css and js files).

Research into user interfaces has concluded that interactions that complete in under 0.1 seconds is ideal as that is considered instantaneous by most users. Under 1 second is the limit of the user's flow of thought to stay uninterrupted during the delay, and 10 seconds tends to be the upper limit to how long a user will wait before giving up on using the app. [9]

As such, our goal is to stay well under the 10 seconds mark for any reasonably fast modern connection. We define reasonably fast as a regular 2G connection with 250mbps throughput and 300ms roundtrip latency. The Chrome developer tools allow us to throttle our connection in order to simulate slower, less than ideal connections, as well as offering us a tool to get a thorough breakdown of our applications loading time.

Below is the non-throttled loading time of our app on UW Place's excellent quality ResNet connection with 100mbps throughput and under 30ms roundtrip latency to Cloudflare's servers.





























Name Path	Me...	Stat... Text	Type	Initiator	Size Conte	Time Laten	Timeline		
 toc-staging.az...	GET	304 Not...	text...	Other	165 B 894 B	32 ... 31 ...			
 initialize.css	GET	200 OK	text...	toc-stagi... Parser	2.6 ... 7.2 ...	41 ... 41 ...			
 es6-module-l... /dependencies	GET	200 OK	app...	toc-stagi... Parser	19... 79...	110... 57 ...			
 system.src.js /dependencies	GET	200 OK	app...	toc-stagi... Parser	18... 71...	226... 225...			
 config.js	GET	304 Not...	app...	toc-stagi... Parser	165 B 2.1 ...	34 ... 34 ...			
 initialize.js	GET	304 Not...	app...	toc-stagi... Parser	165 B 1.3 ...	106... 105...			
 data:applicati...	GET	200 OK	app...	toc-stagi... Parser	0 B 3.9 ...	0 ms 0 ms			
 app.js	GET	200 OK	app...	es6-mo... Script	201... 749...	276... 190...			
 ng-inspector.js aadgmnobpd...	GET	200 OK	app...	inject.js:7 Script	(fro... 1 ms	1 ms 1 ms			
 app.css	GET	200 OK	text...	app.js:10 Script	(fro... 0 ms	0 ms 0 ms			
 data:applicati...	GET	200 OK	app...	Other	0 B 373...	67 ... 67 ...			
 2012345b479... www.gravatar....	GET	304 Not...	ima...	app.js:3 Script	(fro... 0 ms	0 ms 0 ms			
 205e460b479... www.gravatar....	GET	304 Not...	ima...	app.js:3 Script	(fro... 0 ms	0 ms 0 ms			
 205e460b123... www.gravatar....	GET	304 Not...	ima...	app.js:3 Script	(fro... 0 ms	0 ms 0 ms			

Figure 12: Non-throttled loading performance of prototype

In this ideal case, our application loaded completely in well under 1 second, which is a great result as it falls under the 1 second threshold where the user's train of thought can stay uninterrupted.

Below is the list of options offered by the Chrome developer tools for network throttling:

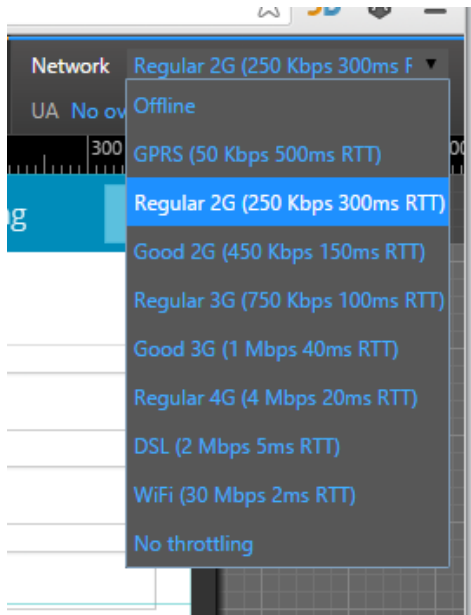


Figure 13: List of network throttling options for Chrome developer tools

We have tested each of these throttling options (except the offline and GPRS options) 5 times and recorded the average results, with the highest and lowest times discarded, in the chart below.

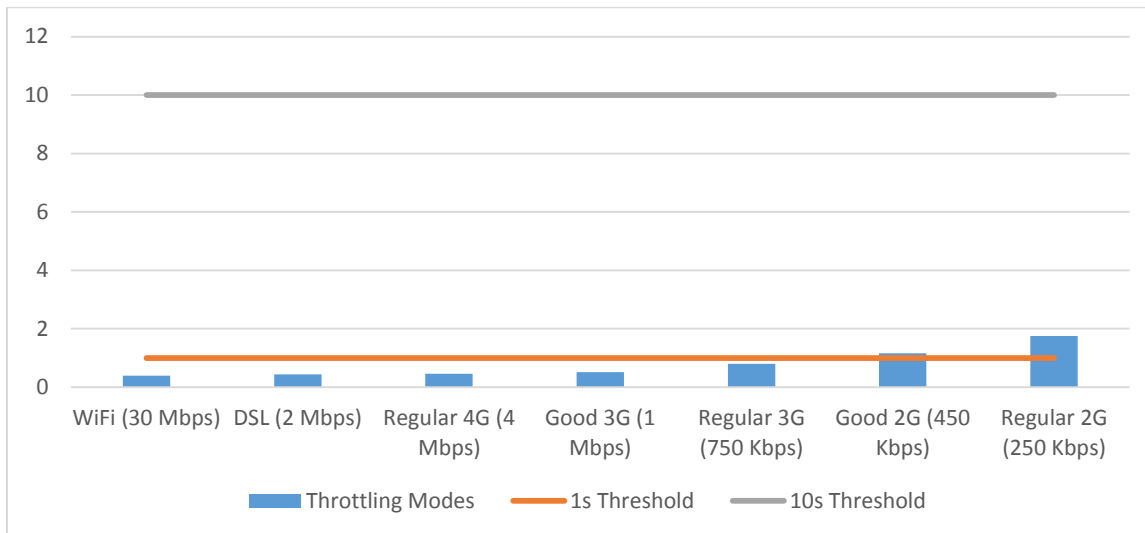


Figure 14: Throttled loading performance of prototype

We can see from the chart above that only the two slowest 2G connections resulted in a loading time of more than 1s. Even then, the slowest 2G connection at 250 Kbps loaded in 1.75s, which is far below the 10s threshold we have set for ourselves. With the prevalence of 3G connections, our application should be able to load in less than 1 second for most users on initial load, and load even faster on subsequent launches due to browser caching mechanisms.

In summary, from both the perspective our subjective testing with the app, as well as from the quantitative measurements above, we can conclude that our user interface meets the performance criteria outlined in our specifications.

4.3 Continuous Integration Test Data

A variety of unit tests were implemented to ensure stability of the messaging system throughout development. Along with unit tests for each component there were also integration tests to make sure the entire application as a whole is always functioning properly. The following tests were performed:

4.3.1 Peer To Peer Messaging

This test evaluates that a message being sent from one peer to the other is reliable and in order. It first sets up 2 peers by generating 2 unique hash names. One peer is designated as the Sender and the other peer is designated as the Receiver. Both peers add each other as contacts, the Sender then sends a specific number of messages to the Receiver. The message itself is the message number count. The test will fail if the Receiver doesn't receive all the messages in the correct order. The pseudo code for this is given below:

Sender:

```
for (var i = 0; i < MESSAGE_COUNT; i++) {  
    send(i, RECEIVER_HASHNAME)  
}
```

Receiver:

```
var count = 0  
while (count < MESSAGE_COUNT) {  
    if (new_message()) {  
        var msg = get_message()  
        expect(msg).toEqual(count);  
        count++;  
    }  
}
```

Note: The implementation of this is slightly different since we rely on callback functions to send and receive messages.

This test currently runs without failure. The message count is currently set to 100 000 messages. Performance metrics was also taken for the sending and receiving of messages. To do this, the test is modified by taking time measurements between messages and maintaining a moving average. Currently, under ideal network conditions the average delay between messages is 70 ms. This test was also set to fail if a message takes longer than 2 seconds which currently does not occur in the latest build.

4.3.2 Peer to Multi-Peer Messaging

This test creates 1 primary sender and a fixed number of receivers. The number of receivers currently set is 10. The primary sender creates a randomly generated string for each receiver and sends that string

to each receiver. The receivers then echo back the same message to the sender. The sender verifies that the echo message is in fact the same. This test also currently runs without failure, and the performance measurements also show an average delay of about 70 ms.

These 2 integration tests are used to verify our first essential specification, the messages sent between peers is reliable and robust with no lost messages and correct message ordering. The performance recording for the average message delay was automated and performed under multiple trials. The following graph shows the average delay under 250 trial runs:

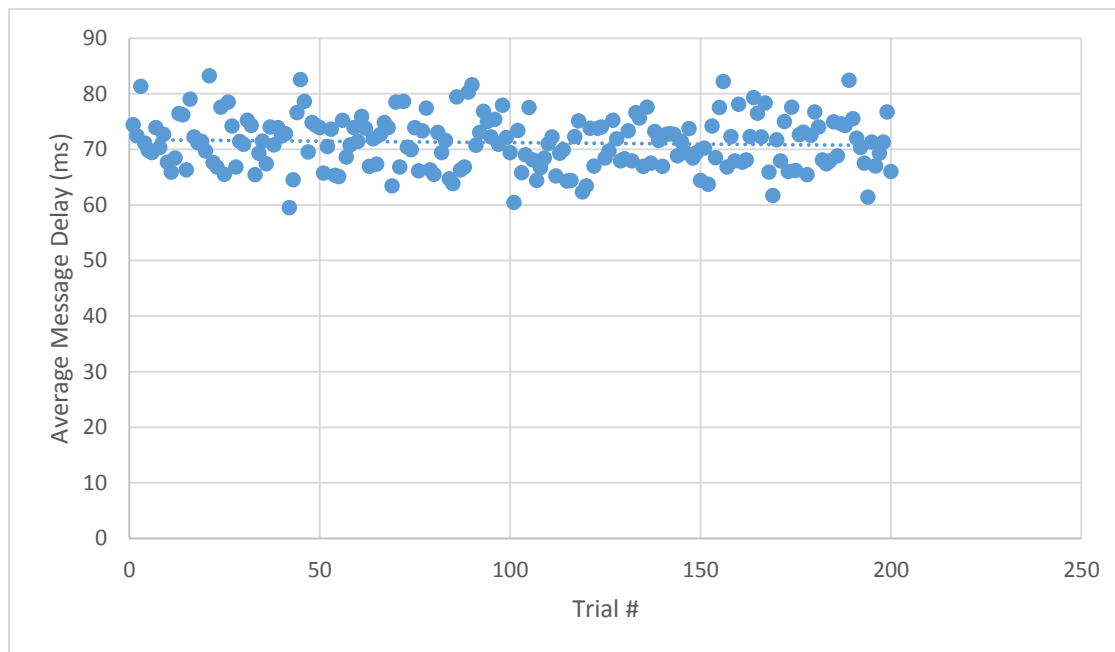


Figure 15: Average message delay for prototype

As can be seen the message delay is slightly above 70 ms and the maximum average delay does not exceed 83 ms. None of our messages exceeded a 2 second delay as specified in the first functional requirement.

4.4 Resource Utilization Efficiency Data

The non-functional design specification that was indicated in section 3 of the report was efficiency. Not only the product must be functional, the performance and efficiency of the product needs to be met a certain standard. It is extremely important to ensure the performance and efficiency of the product does not degrade the user's experience. Therefore, we have included the criteria, efficiency of performance to be an essential non functional specifications. The efficiency and performance of an application is referred to the bandwidth consumption of the application and the latency of the application's events.

In order to satisfy or verify this essential non functional specifications, experiments are targeted on the network memory usage during the launch of the application and sending/receiving messages. Therefore, these experimental data were exacted from the usage of the Chrome Developer Tools. Inside Chrome Developer Tools, there consist a Network monitor category that monitors the amount of data being sent

and received. In addition it provides the details of the time duration for an every event in which we can calculate the total time for the application to be launched.

Table 5: Bandwidth usage measurements for prototype

Event	Bandwidth Usage (KB)	Latency (s)
Application Launch	5.2	1.30
Send/Receive Message (1000 characters)	2.4	0.57

From the Chrome Developer Tools, measurements for the memory consumption and time duration showed that 5.2 KB were transferred in 1.30 seconds on the client side when launching the application. Ideally, having less data consuming the bandwidth and shorter time run in launching the application would be beneficial to some extent. The latest prototype has already refactored and optimized to the level where the bandwidth consumption becomes insignificant. While the latency in launching the application is measured at 1.30 seconds where this can be possibly improved on. However, the current latency of 1.30 seconds can be considered to be insignificant enough in which it will not have a heavy influence to the user's experience.

The user experience is most heavily affected by the usage of the application. Since the application's purpose is for communicating with other clients, sending and receiving messages is a huge factor in the user's experience. When sending or receiving a single message (1000 characters), 2.40 KB are transferred over the network and it took about 0.57 seconds under experimental environment. Stated in the essential non functional specification, the required limit to not exceed 10KB with 1000 characters being sent through the network. From our experimental run, the bandwidth consumption is 20% of the maximum bandwidth consumption limit. Therefore, it is verified that the application meets the essential non function specification, efficiency.

5. Discussion and Conclusion

5.1 Evaluation of Final Design

From our perspective, our final design satisfies all essential specifications. After investigating which technologies that will be used in our project, we have carefully chosen the correct technologies to fulfill the essential specifications. Our implementation of these technologies were robust enough that every essential specification can be demonstrated to have been met in our prototype.

We believe we have achieved our objective of creating a messaging application that safeguards user privacy as our utmost concern. Our messaging application indeed does not rely on any external servers to communicate with other clients, and messages are encrypted from end-to-end as they are delivered from one user to another. We have integrated the ability to save encrypted copies of message history on the local device, as well as having it synchronized to a remote storage location in a secure manner to be replicated across devices.

However, due mostly to time constraints, we were not able to satisfy all the non-essential specifications. Some of the performance oriented specifications were calibrated a bit too optimistically when we first built the specifications document, and ended up being not realistic given the constraints of a real world implementation. Some of the “nice-to-have” features listed in our non-essential specs also did not get implemented because we chose to instead to focus on fine-tuning the performance and reliability of our existing implementation. With enough additional time, we could most likely still implement these features on top of our existing prototype.

5.2 Use of Advanced Knowledge

Our project and design comprises multiple components which requires the knowledge of different upper-year engineering knowledge. Working with distributed systems, encrypting messages over the network, and implementing software architectural pattern are considered upper-year engineering knowledge which are used in our design.

Distinctively, our project major component is to operate in a decentralized system. Therefore, our design will be bounded by the knowledge of distributed systems in order to achieve a decentralized system. Each peer or machine will operate as an independent entity or node in the system thus, dynamic hash tables will be required to allow communication between each node. TeleHash DHT, an open source project which we can include into our design allows us to achieve this communication. The idea of using decentralized system originated from the course ECE 454 – Distributed Systems which we were all enrolled in during our previous academic study term.

The involvement of encryption as a upper-year engineering knowledge was because our project heavily emphasizes on the user privacy policy. Encryption is the major component of our system to allow the user privacy policy to be successful. For each message that will be sent over the network it will be encrypted to prevent tampering and eavesdropping. TeleHash DHT enable us to achieve the encryption when the messages are sent over the network.

Besides features and functionalities, we have decided to organize and implement the design in a more efficient way. The software architecture of our design will include design patterns to enhance its

performance and readability. To follow a good software coding standards, we have included AngularJS, a JavaScript framework in to our project to allow the usage of MVC (Model-View-Controller). MVC is a software architecture pattern for implementing the user interface by separating the internal levels of representing information.

5.3 Creativity, Novelty, Elegance

Distributed, client-side only applications are often thought of as crippled in some way due to the absence of a central server in providing certain functionality. We believe the most novel aspect of our design is in our ability to achieve feature parity with most existing instant messaging clients despite our distributed architecture.

We accomplished this through a creative combination of third party libraries, open protocols, and our own custom logic. We implemented contact discovery services by querying a DHT consisting of our online users and any other users of Telehash applications. We implemented simultaneous login by augmenting Telehash's identity model with our own user identity model, consisting of multiple cooperative Telehash ID's engaging in leader elections in accordance with the Raft distributed leader election algorithm. We implemented user data persistence across devices through the integration of the remotestorage.io protocol in our application, and allowing users to choose their own storage provider or even self-host their own storage, while at the same time allowing them the option to keep their data completely locally if they so prefer.

We believe the combination of our distributed architecture and the convenience features normally only seen in client-server applications is something that hasn't been done before and will become one of our application's major differentiators in the market.

Another major novel aspect of our application is our use of web technologies to deliver a truly universally usable application. We have native clients for almost all mobile OS's due to our integration with Apache Cordova, as well as almost all desktop platforms due to our integration with node-WebKit. We are able to deliver all of these native applications from a single unified codebase written in HTML, CSS and JavaScript, with only minimum configuration files necessary to account for platform specific settings. And for any platform that we don't offer a native client, our application can be accessed through the web browser in a fully functional manner, either through our own hosted version or downloaded to run locally or on a personal server.

5.4 Quality of Risk Assessment

Back in our ECE498A report, the list of specification and risk assessment were discussed. There were three major risks that could have hindered the project's success.

Firstly, the quality, stability, interoperability and maturity of open-source libraries needed for certain project components was the biggest risk or obstacle to our project's success. The project depended on a number of different open source libraries for operation (encryption, peer discovery, online data storage, and etc). These open source libraries are under heavy development which can potentially be incomplete or possibly non bug free functionalities. Therefore, the final product of project cannot guarantee the full functionality as expected in the design. Furthermore, compatibility is another factor or risk that

influences the success of the project. Unable to fully connect or produce an end to end result of the project due to compatibility and result in a unusable product.

This first risk inhibits our ability to increase the success of the project because the open-source libraries are developed by external developer in which we have limited control over. For any discovered issues with any open source libraries that is originated from the library itself, it is expected that a workaround involves using an inelegant solution. Even so, there are cases where no solution can be feasible to tackle the problem, thus it can result in replacing the library which can consume a large amount of time in searching, implementing, and testing the new library.

This risk was realized many times throughout development where we found issues in the tools, frameworks and libraries we were working with. In most cases we mitigated these issues by submitting bug reports to the project pages of the library, and luckily in most of these cases, the project maintainers would then promptly address the issues we raised or explain why the behavior is as expected. In the cases where the issues could not be satisfactorily resolved, we had to resort to our own approaches to resolve the issue.

One instance of this was the limitation with the Telehash library in being unable to handle multiple sign-on. To resolve this, we refactored our user identity models to consist of multiple Telehash id's abstracted behind a single identity, and having each Telehash instance communicate with each other to coordinate message sending and reception. There were also issues that forced us to reassess our choice in technologies. For instance, limitations of the Snap.js library on mobile interfaces resulted in our switching to a different framework for our app layout.

Secondly, failure to meet time constraints for the project's deliverables was our next major risk. Time management is an important factor in determining the completion of project before the deadline because we need to invest enough time to complete each component of the project. Our project treads on many uncharted territories in the field of instant messaging and requires knowledge of several advanced networking topics, so time required for completing each component can be difficult to estimate.

The risk of failing to meet the time constraints for the project's deliverables has not occurred throughout the development of this project. Throughout the course of the design project, we have successfully met and delivered the required documentations and demonstrated our product successfully before the deadline.

Our success in avoiding the failure to meet time constraints for the project's deliverables is scheduling occurring meetings to work on the project. Before each deliverable, we discussed and planned what is needed to be done, and then distribute the work among the team members. This is normally done for the written documentation however, for the development of the project there was a leader who guided the rest of the team members. The leader of the group ensured that there was enough time to develop the project before the deadline. The leader has the entire knowledge of the project's architecture and knew what was required to be completed. He has scheduled the meetings for us which fits our time schedule. In addition, our time schedule throughout fourth year was flexible enough to invest a good

amount of time to work on the design project each week. The choices of our course heavily influenced the avoidance of this risk, failure to meet time constraints for the project's deliverable. Our time management and organization skills played an enormous role in successfully delivering the deliverables before the deadlines.

Other than the time management and organization skills, the judgment calls and prioritizing the development of different features of the project was another contributing factor in successfully meeting the deadlines. There are moments and cases where certain features will consume an enormous time to be implemented and developed. These features are normally non-essential feature but for the cases of essential features, group discussion were required to determine our next steps. Normally, more time and effort is invested to complete the essential feature without delaying the other essential features. However, when time is not luxurious, we resorted to developing temporary solution which is inelegant and quicker implement.

Lastly, the possibility of team leaving or disbandment was our last major risk in affecting the success of the project. In every project involving two or more people, some degree of conflict will be inevitable. Members may decide to leave the team in the worst case. Also, there is a chance that certain members may need to leave the program in the second term due to falling out or other special circumstances. In the situation where a team member were to leave the group, it will greatly affect the project and the remaining team members. Addition time required to cover for the remaining work of the missing member, and the morale of the team will be heavily affected. In the worst case, the entire team can disband and abandon the project.

Fortunately, until now there has not been a case where a member had to disband from the team. However, our team was temporary inactive for our coop term since our coop placement was in different location and it was difficult to arrange or schedule a perfect time meeting to continue on the project's development. Nonetheless, we have also concentrated in our academic work to ensure that we're able to pass and advance to the next academic term together.

We found out that coordinating work on the project during the co-op term was extremely difficult. We ended up overcoming this by using the work term to individually study the technologies we were using in the project and discovering new alternative technologies, and then focusing on actual implementation once school started again in January. We even decided to live in the same unit in UW Place to remove scheduling and coordination overhead that we experienced last term and improve our efficiency. This last decision contributed greatly to our efficiency and productivity since the beginning of the 4B term.

5.5 Student Workload

Table 6: Table illustrating the number of hours invested for each student

	Asif Arman	SangHoon Lee	Qi Liu	Danny Yan
Meetings and Brainstorming	5	5	5	5
Finding Consultant	1	1	1	1
Project Abstract	1	1	1	1

Project Specification & Risk Assessment Document	3	5	4	3
Research	29	27	50	18
Implementation	52	55	64	39
Final Report	1	2	3	4
Student Work	91	96	128	71
Student Work Divided Total Group Work Percentage	23.58%	24.87%	33.16%	18.39%

The distribution of the work for the design project is shown in the table below. Throughout the course of the project's development, Qi Liu took the initiative to become the leader of the project. Qi Liu invested an enormous time in researching the best technologies to be used in the project's design. There were scenarios where multiple candidates were compared to determine the best solution to be implemented. In addition, Qi Liu has guided the entire team in the development where he has prioritized the right feature to be developed and handle the time management in scheduling group meetings.

The implementation work was led by Qi Liu however, the other members also took part in the implementation. In the scheduled group meeting, each member will take turn in implementing the design. Therefore, there will be one developer with three observers. The developer is the person who implements the design while the other three observers verify and discuss on the design of the implementations. This was similar to pair programming except there are four members instead of two. This can be considered inefficient however, we believe that it is the easiest way to synchronize the entire team in development of the project.

From the above table, we can also conclude that Danny Yan contributed a unevenly and least amount of work. The reason for the significant difference between Danny Yan and the rest of the team members is due to his external activities and lack of attendance during group meetings. Danny Yan has attended most of the group meetings but there are some group meetings which conflicts with his schedule. He had to be unavailable and attend his non disclosed family meetings back at home. His lack of attendance to few of the group meetings affected his synchronization with the team. Therefore, he was less likely to be aware of the current progress and it influenced his ability to implement or develop during the next group meetings.

References

- [1] D. Goodin, "Skype replaces P2P supernodes with Linux boxes hosted by Microsoft (updated)," 01 May 2012. [Online]. Available: <http://arstechnica.com/business/2012/05/skype-replaces-p2p-supernodes-with-linux-boxes-hosted-by-microsoft/>. [Accessed 01 June 2014].
- [2] Microsoft, "Skype Privacy Policy," 2014. [Online]. Available: <http://www.skype.com/en/legal/privacy/>. [Accessed 01 June 2014].
- [3] Tor Project, "Tor Project," Tor Project, 2014. [Online]. Available: <https://www.torproject.org/>. [Accessed 01 July 2014].
- [4] J. Miller, "Telehash," 2014. [Online]. Available: <http://telehash.org/>. [Accessed 01 July 2014].
- [5] Adobe Systems, "PhoneGap," Adobe Systems, 2014. [Online]. Available: <http://phonegap.com/>. [Accessed 01 July 2014].
- [6] Dropbox Inc, "Dropbox," 2014. [Online]. Available: <https://www.dropbox.com/developers/datastore>. [Accessed 01 July 2014].
- [7] Google Inc, "Google Drive SDK," 2014. [Online]. Available: <https://developers.google.com/drive/web/about-sdk>. [Accessed 01 July 2014].
- [8] remoteStorage Contributors, "remoteStorage - An open protocol for per-user storage," 2014. [Online]. Available: <https://remotestorage.io/>. [Accessed 9 February 2015].
- [9] Stanford University, "SJCL," 2014. [Online]. Available: <http://bitwiseshiftleft.github.io/sjcl/>. [Accessed 01 July 2014].
- [10] J. Nielsen, "Response Times: The 3 Important Limits," 01 January 1993. [Online]. Available: <http://www.nngroup.com/articles/response-times-3-important-limits/>. [Accessed 21 December 2014].
- [11] T. Simonite, "New Scientist Blogs," 07 December 2007. [Online]. Available: <http://www.newscientist.com/blog/technology/2007/12/instant-message-irrelevance.html>. [Accessed 01 June 2014].

Appendix A: Completed Prototype Hazard Disclosure Form

ECE498B: Prototype Hazard Disclosure Form*		Group number: <u>2015.010</u>
<p><i>Instructions:</i> Answer all the following questions by putting an <i>X</i> in either the <u>yes</u> or <u>no</u> column. If unsure, answer <u>yes</u>. If you answer <u>yes</u> to any question, set up an appointment with the Lab Instructor to ensure your prototype is safe for the symposium. Include this completed form in your Final Report (Appendix A) even if you answer <u>no</u> to all questions.</p>		
Question: Does your prototype...	yes	no
1. include any circuitry that you designed or built by yourself?		x
2. include any circuitry that is not enclosed in an approved plastic or metal electrical box?		x
3. involve any 120V AC circuitry/device that is not approved by CSA or ESA?		X
4. involve circuitry that is not connected to its power supply with a fuse and a switch?		X
5. use high-capacity or high-density (e.g., lithium-ion) batteries?		X
6. emit non-trivial amounts of RF radiation?		X
7. involve x-rays or radioactive materials?		X
8. use unprotected lasers of any class?		X
9. involve strobe lights?		X
10. have exposed moving parts that may pinch, hit, or crush a person?		X
11. involve projectiles or any part that can fly?		X
12. have exposed sharp edges or points?		X
13. use high-pressure gases or liquids?		X
14. involve irritating or dangerous chemicals (assume all nano-materials are dangerous)?		X
15. involve any biological materials (dead or alive) or any food/drink?		X
16. emit dangerously loud sounds?		X
17. eject gas, particles, or fluids into the environment?		X
18. involve accessible components that reach temperatures above 40°C or below 0°C?		X
19. involve any other hazard? Describe:		X
<p>Lab Instructor Inspection Report (needed only if "yes" is checked to any question above)</p> <p><input type="checkbox"/> Prototype is deemed to be <u>unsafe</u> in its current state and requires another inspection after changes are made.</p> <p><input type="checkbox"/> Prototype is deemed to be <u>conditionally acceptable</u>[†] for the symposium, and another inspection is not required.</p> <p><input type="checkbox"/> Prototype is deemed to be <u>safe</u> for the symposium in its current state.</p> <p>[†]State on the back of this form what minor changes are required for the prototype to be considered safe for the symposium.</p> <p>Lab Instructor Signature _____ Date _____</p>		
<p>Project Consultant Inspection Report at Final Prototype Demonstration (needed for all projects)</p> <p>yes no</p> <p><input type="checkbox"/> <input type="checkbox"/> The group appears to have accurately answered the above 19 questions.</p> <p><input type="checkbox"/> <input type="checkbox"/> The prototype presently appears to be safe for the public symposium.</p> <p>Consultant Signature _____ Date _____</p>		

Appendix B: Completed Symposium Floor Plan Request Form

ECE498B: Symposium Floor Plan Request Form*		Group number:2015.010	
<p>Instructions: Answer all the following questions by putting an X in either the <u>yes</u> or <u>no</u> column. Provide additional information in the far right column as requested. Include the completed form in Appendix B of your Final Report, even if you answer <u>no</u> to all questions. Requests for equipment (e.g., an oscilloscope, a power supply, a large monitor, a computer, or a lab stool) should <u>not</u> be included on this form; instead, for such equipment requests use the on-line reservation system as described in the <i>Symposium Checklist and Schedule</i> document.</p>			
Question:	yes	no	if you answered "yes"...
1. Does your project require one or more hardwire <u>internet connections</u> at the symposium? Note: We provide Ethernet connections only, via a male RJ-45 connector. The connection comprises a static IP address in the uwaterloo.ca domain on a shared 100Mbps link. Do not count on the DC building wireless system being available for your project.		x	If yes, state how many connections you require:
2. Do you desire to use your <u>own wireless router</u> at the symposium? If yes, you will need to have your setup approved well before the symposium date. Unapproved routers are strictly prohibited.		x	If yes, contact Paul Ludwig for details as soon as possible.
3. Each booth has a 7.5-amp 6-outlet power bar. Does your project require <u>more than 7.5 amps</u> of mains (120 V AC) power?		x	If yes, state how many amps you require in total:
4. Each booth has a 7.5-amp 6-outlet power bar. Does your project require <u>more than 6 outlets</u> ?		x	If yes, we will supply your booth with two power bars.
5. Do you intend to put any <u>electronics/computers on the floor</u> under your booth?		x	If yes, we will ensure you can do this safely.
6. Does your project involve <u>projectiles or flying parts</u> ? (We have a large "cage" at the symposium for projects that involve projectiles or flying parts.)		x	If yes, state the nature of the projectile or flying part:
7. Does your project require <u>special lighting</u> conditions (e.g., dim light, bright light)? We will do our best to accommodate lighting requests, but no guarantees are made.		x	If yes, state the nature of the desired lighting conditions:
8. The default booth consists of a 5' wide by 2.5' deep table. The table is 2.54' tall. Does your project require <u>extra table space</u> (giving you a total table area of 7.5' wide by 2.5' deep)? We will do our best to accommodate space requests, subject to the urgency of the request and overall space and safety constraints.		x	If yes, explain specifically why you are requesting the extra table space:
9. The default floor space, including the table and area for you/visitors to stand, is around 6' by 6'. Does your project require <u>extra floor space</u> ? We will do our best to accommodate space requests, subject to the urgency of the request and overall space and safety constraints. Note: Due to the extra-large class size this year, very few requests for extra floor space will be granted.		x	If yes, explain why you need the extra floor space: State how much extra space you are requesting:
10. Do you have any other special floor plan requests?		x	State your request: