# ENHANCING ZOTERO, RESEARCH MANAGEMENT TOOL WITH P2P CAPABILITIES

**PROJECT REPORT**

**DESIGN ORIENTED PROJECT (IS F376)**

SHIKHAR VASHISTH 2012C6PS436P

**BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI**

# ACKNOWLEDGEMENT

I am thankful to computer science faculties who have given me a chance to do Design Project. I would like to thank Prof. Haribabu K for his guidance and constant supervision as well as for providing necessary information regarding the project & also for his support in the progress of the project. I would like to express my gratitude towards my fellow colleague Yash Sinha who has willingly helped me whenever I was stuck up in my design project. At last I would like to thank each and every one who has supported me by any means in the progress of this design project.

# TABLE OF CONTENTS

# 1. INTRODUCTION

Zotero is a research tool for managing online references. Developed by the Centre for History and New Media at George Mason University, Zotero is a Firefox extension that provides users with automated access to bibliographic information for resources viewed online. Using “translators” for several hundred websites, online databases, and commercial sites such as Amazon.com, Zotero “senses” the bibliographic information contained in a web page and—when the user clicks the Zotero icon—gathers that information and places it in the user’s library of sources. In this way, online researchers can quickly and easily gather the information they will need later to review and cite references and create bibliographies. Zotero includes features to manage sources, and users can also manually enter sources. The result is a centralized location for gathering and storing references, significantly streamlining the research process.

The translators that allow Zotero to access web-based bibliographic information have been written for institutional libraries, the Library of Congress, databases such as LexisNexis, archiving services such as JSTOR, newspapers, and hundreds of other organizations around the world. So far, more than 100 colleges, universities, and other organizations recommend Zotero to their constituents, and nearly 30 of these provide tutorials or workshops on how to use the tool. Zotero is equally useful for undergraduates, graduate students, faculty, and other researchers—anyone who uses the Internet for research and needs a simple way to organize references.

Zotero even allows to share our data with other users. One can upload his/her data on Zotero server and can share it with other users but Zotero server imposes some limit on amount of data we can upload. So if one only wants to share data among people who are on the same network then uploading data to Zotero server and downloading it from there by others makes no sense. This project is a step to eliminate this problem and other similar problems by allowing data to get shared through peer to peer network. We are planning to develop a Firefox Extension for Zotero that will enable it to share data through peer to peer network. We have used Chord protocol for peer-to-peer distributed has tables that allows nodes to connect in O(log(n)) time thus improving the performance of whole system.

# 2. PEER-TO-PEER NETWORKS

Peer-to-peer (P2P) computing or networking is a distributed application architecture that partitions tasks or workloads between peers. Peers are equally privileged, equipotent participants in the application. They are said to form a peer-to-peer network of nodes.

Peers make a portion of their resources, such as processing power, disk storage or network bandwidth, directly available to other network participants, without the need for central coordination by servers or stable hosts. Peers are both suppliers and consumers of resources, in contrast to the traditional client-server model in which the consumption and supply of resources is divided. Emerging collaborative P2P systems are going beyond the era of peers doing similar things while sharing resources, and are looking for diverse peers that can bring in unique resources and capabilities to a virtual community thereby empowering it to engage in greater tasks beyond those that can be accomplished by individual peers, yet that are beneficial to all the peers.

While P2P systems had previously been used in many application domains, the architecture was popularized by the file sharing system Napster, originally released in 1999. The concept has inspired new structures and philosophies in many areas of human interaction. In such social contexts, peer-to-peer as a meme refers to the egalitarian social networking that has emerged throughout society, enabled by Internet technologies in general.

In P2P networks, clients both provide and use resources. This means that unlike client-server systems, the content serving capacity of peer-to-peer networks can actually increase as more users begin to access the content (especially with protocols such as Bit torrent that require users to share). This property is one of the major advantages of using P2P networks because it makes the setup and running costs very small for the original content distributor.

# 3. CHORD-DHT

## 3.1. DEFINITION:

Chord is a protocol and algorithm for a peer-to-peer distributed hash table. A distributed hash table stores key-value pairs by assigning keys to different computers known as "nodes", a node will store the values for all the keys for which it is responsible. Chord specifies how keys are assigned to nodes, and how a node can discover the value for a given key by first locating the node responsible for that key. Chord is one of the four original distributed hash table protocols, along with CAN, Tapestry, and Pastry. It was introduced in 2001 by Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan, and was developed at MIT.

Peer-to-peer systems and applications are distributed systems without any centralized control or hierarchical organization, where the software running at each node is equivalent in functionality. A review of the features of recent peer-to-peer applications yields a long list: redundant storage, permanence, selection of nearby servers, anonymity, search, authentication, and hierarchical naming. Despite this rich set of features, the core operation in most peer-to-peer systems is efficient location of data items.

Chord supports this operation of mapping a key onto a node efficiently in O (log (n)) time. Chord uses a variant of consistent hashing to assign keys to Chord nodes. Consistent hashing tends to balance load, since each node receives roughly the same number of keys, and involves relatively little movement of keys when nodes join and leave the system.

In Chord each node keeps some routing information in form of routing table in which it maintains information about log (n) nodes of the network and keeps updating it as nodes joins and leaves. For each lookup operation system makes use of this information to search for a node in O (log (n)) time.

## 3.2. CHORD OPERATIONS:

**Create Node**

```
n.create()
  predecessor = nil;
  successor = n;
```

**Node Join:**

```
n.join(n')
  predecessor = nil;
  successor = n'.find_successor(n);
```

**Node Stablize:**

```
n.stabilize()
  x = successor.predecessor;
  if (x\in(n, successor))
    successor = x;
  successor.notify(n);
```

**Notify:**

```
n.notify(n')
  if (predecessor is nil or n'∈(predecessor, n))
    predecessor = n';
```

**Fix fingers:**

```
n.fix_fingers()
  next = next + 1;
  if (next > m)
    next = 1;
  finger[next] = find_successor(n+$2^{next-1}$);
```

**Find Successor:**

```
n.find_successor(id)
  if (id ∈ (n, successor] )
    return successor;
  else
    n0 = closest_preceding_node(id);
    return n0.find_successor(id);
```

**Closest preceding node:**

```
n.closest_preceding_node(id)
  for i = m downto 1
    if (finger[i]∈(n,id))
      return finger[i];
  return n;
```

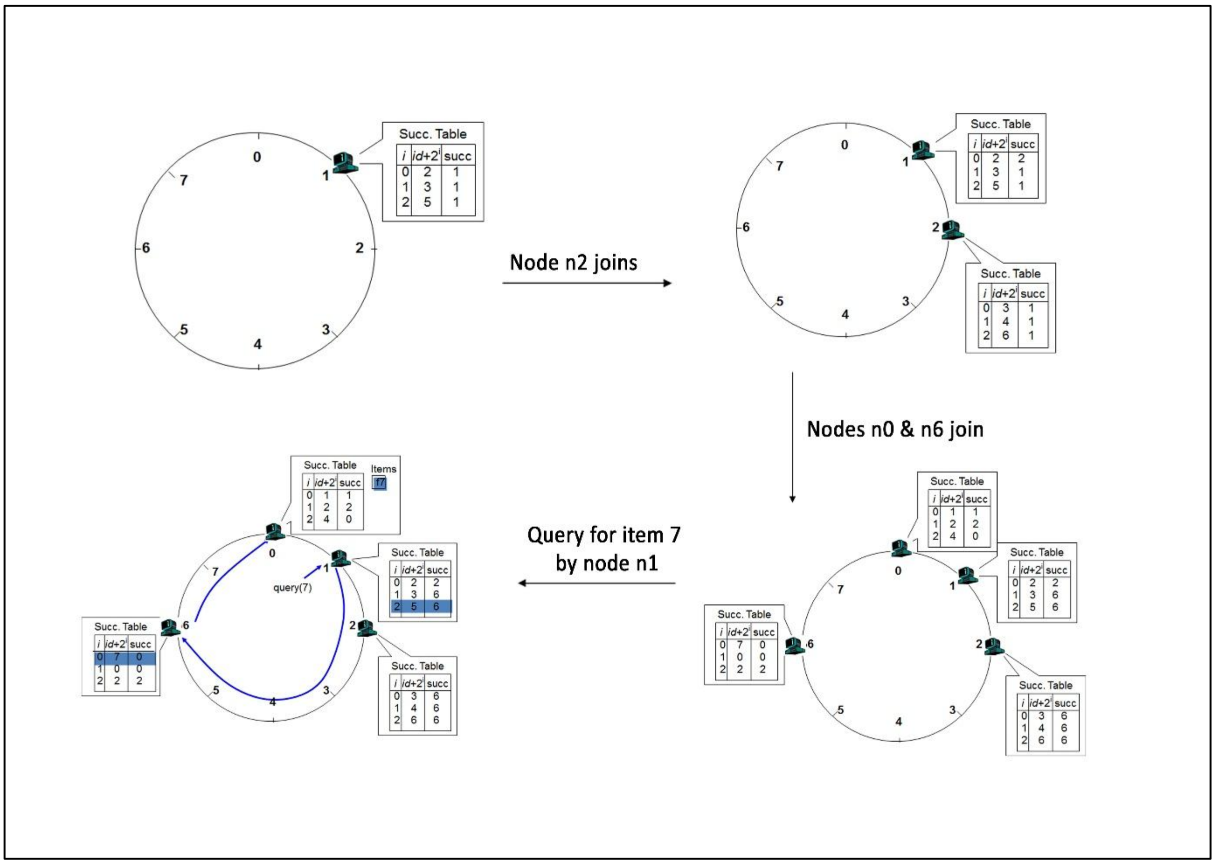

*Figure 1- Demonstrating node join and lookup operation in Chord*

# 4. FIREFOX ADDON DEVELOPMENT

## 4.1 WHAT ARE ADDONS?

Add-ons are installable enhancements to the Mozilla Foundation's projects, and projects based on them. Add-ons allow the user to add or augment application features, use themes to their liking, and handle new types of content. Things like fine-grained tab controls, mouse gestures, extensive toolbars and buttons, a feed reader, integration with a variety of web applications, or sophisticated tools to assist with web design all are provided by add-ons.

There are two main types of add-on: Extensions add new features to the application, while Themes modify the application's user interface.

**Extensions:** Extensions can be used to modify the behaviour of existing features to the application or add entirely new features. Extensions are especially popular with Firefox, because Mozilla developers intend for the browser to be a fairly minimalistic application in order to reduce software bloat and bugs, while retaining a high degree of extensibility, so that individual users can add the features that they prefer.

**Themes:** Themes are add-ons that customize the application's user interface. There are two sorts of themes: lightweight themes and complete themes.

## 4.2. TECHNOLOGIES USED IN DEVELOPING EXTENSION

- XPCOM (Cross-Platform Component Object Model)
- XUL (XML User Interface Language)
- JavaScript
- CSS (Cascading Style Sheets)

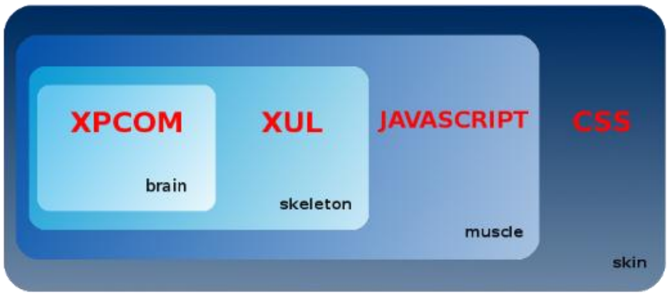

*Figure 2- Technologies in extension development*

*Figure 3- Technologies in extension development*

## 4.3. XUL (Extensible Mark-up Language)

XUL (pronounced as zool), which stands for XML User Interface Language, is a user interface markup language that is developed by Mozilla. XUL is implemented as an XML dialect; it allows for graphical user interfaces to be written in a similar manner to Web pages.

XUL can be used to write cross-platform applications such as Mozilla Firefox, where it is interpreted by the layout engine known as Gecko which renders Firefox's user interface and Web page display. XUL has all the advantages of other XML languages. For example XHTML or other XML languages such as MathML or SVG can be inserted within it. Also, text displayed with XUL is easily localizable, which means that it can e translated into other languages with little effort.

Programmers typically define a XUL interface as three discrete sets of components:

- content: the XUL document(s), whose elements define the layout of the user interface
- skin: the CSS and image files, which define the appearance of an application
- locale: the files containing user-visible strings for easy software localization

XUL provides the ability to create most elements found in modern graphical interfaces. Some elements that can be created are:

- Input controls such as textboxes and checkboxes
- Toolbars with buttons or other content
- Menus on a menu bar or popup menus
- Tabbed dialogs
- Trees for hierarchical or tabular information
- Keyboard shortcuts

## 4.4. XPCOM (Cross Platform Component Object Model)

XPCOM (Cross Platform Component Object Model) is a cross-platform component model from Mozilla. It is similar to Microsoft COM and CORBA. It features multiple language bindings and IDL descriptions thus programmers can plug their custom functionality into the framework and connect it with other components.

Mozilla is constructed from a collection of components, each of which performs a certain task. For example, there is a component for each menu, button and element. The components are constructed from a number of definitions called interfaces. An interface in Mozilla is a definition of a set of functionality that could be implemented by components. Components are what implement the code in Mozilla that does things. Each component implements the functionality as described by interfaces. A single component might implement multiple interfaces. And multiple components might implement the same interface. Windows and Macintosh versions of a file component would be significantly different. However, they would both implement the same interface. Thus, we can use a component by accessing it using the functions we know from the interface.

XPCOM components are typically implemented natively, which means that they generally do things that JavaScript cannot do itself. However, there is a way in which you can call them, which we will see shortly. We can call any of the functions provided by the component as described by the interfaces it implements. For example, once we have a component, we can check if it implements nsISound, and, if so, we can play sound through it. The process of calling XPCOM from a script is called XPConnect, which is a layer which translates script objects into native objects.

It provides the following features for the cross-platform software developer:

- Component management
- File abstraction
- Object message passing
- Memory management

## 4.5. JavaScript

JavaScript (JS) is a dynamic computer programming language. It is most commonly used as part of web browsers, whose implementations allow client-side scripts to interact with the user, control the browser, communicate asynchronously, and alter the document content that is displayed. It is also being used in server-side network programming (with Node.js), game development and the creation of desktop and mobile applications.

JavaScript is a scripting language first developed in the 1990s, at which time it was created as a way to add dynamic features to web pages. Because it was often used at first to display pop-up windows, marching text in status bars, or in other ways that made web pages less useful to users, the language acquired a reputation as having little practical use and lacking in functionality. The rise of web services like Google Maps, which used JavaScript and asynchronous communications, created an awareness of a set of technologies nicknamed AJAX (Asynchronous JavaScript and XML); that plus the advent of a number of libraries that paper over implementation differences between different web browsers has made it popular.

## 4.6. CSS (Cascading Style Sheets)

Like XML, Cascading Style Sheets (CSS) is a technical specification established by the W3C; it is a style-description language defining the display of data marked up in XML and HTML. As shown in Listing 1, it has an extremely simple syntax. By separating the structure of the data, expressed through HTML or XML, and the display style, indicated by CSS, data can be reused better than it is when structural and stylistic markup are both embedded in HTML.

There are three CSS specifications (Level 1 through Level 3), with progressively powerful features. The Gecko rendering engine handles nearly all of CSS Level 2 and some of CSS Level 3.

# 5. APPENDIX

## 5.1 CHORD IMPLEMENTATION

### MAIN.JS

```
/**********************CHORD INITIALIZED**********************/

var chord = new Chord();

var KEY_LENGTH=5;
var MOD = Math.pow(2,KEY_LENGTH);

var nodeAdd = [1,2,6,15,16,24];

/**********************CREATING NODES**********************/
for (var i = 0; i < nodeAdd.length; i++) {
        var nodeKey = nodeAdd[i];
        // console.log(nodeKey+"=>");
        chord.createNode("node"+nodeKey,nodeKey)
};

console.log(chord.nodeMap.length() + " nodes created");

for (var i = 0; i < chord.nodeMap.length(); i++) {
        console.log(chord.getSortedNode(i));
};

/*****************JOINING AND STABILIZING NODES****************/
for (var i = 1; i < chord.nodeList.length; i++) {
        var node = chord.nodeList[i];
        node.join(chord.getNode(0));
        var preceding = node.successor.predecessor;
        node.stabilize();

        if(preceding == null){
                node.successor.stabilize();
        }
        else{
                preceding.stabilize();
        }
};

console.log("\n************NODES HAVE JOINED AND ARE STABILIZED *******************\n");

chord.printNodes();

/*****************FIXING FINGER TABLES****************/

for (var i = 0; i < nodeAdd.length; i++) {
        var node = chord.nodeList[i];
        node.fixFingers();
};
```

```
console.log("\n***********NODES FINGER TABLES HAVE BEEN FIXED ******************\n");

chord.printNodes();

/******************CREATE DATA OBJECTS*****************/

var dataList = [[1,"Data1 value"],[2,"Data2 value"],[3,"Data3 value"],[4,"Data4 value"]];
var dataObjList = [];

for (var i = 0; i < dataList.length; i++) {
	var dataId = "Data:" + dataList[i][0];
	var dataValue = dataList[i][1];

	dataObjList.push(new Data(dataId,dataValue));
}

for (var i = 0; i < dataObjList.length; i++) {
	console.log(dataObjList[i]);
};

/******************STORING DATA OBJECTS*****************/
for (var i = 0; i < dataObjList.length; i++) {
	dataObjList[i].store(getBootStrap());
};


/******************LOOKUP DATA OBJECTS*****************/
for (var i = 0; i < dataList.length; i++) {
	var dataId = "Data:" + dataList[i][0];
	console.log(getBootStrap().lookUp(dataId))
};

/**********************MAIN ENDS*********************/
```

## CHORD.JS

```
/***************************CHORD CONSTRUCTOR*********************/

function Chord(){
	this.nodeList = [];
	this.nodeMap = new SortedMap();
}

/********************CREATE NODE***********************************/

Chord.prototype.createNode = function(nodeId,nodeKey){
	var node = new ChordNode(nodeId,nodeKey);

	if(this.nodeMap.get(nodeKey) != null)
		console.log("Duplicate Key!!");

	else{
		this.nodeList.push(node);
		this.nodeMap.put(nodeKey,node);
	}
}
```

```
/********************GETTER METHODS***********************************/

Chord.prototype.getNode = function(i){
        return this.nodeList[i];
}

Chord.prototype.getSortedNode = function(i){
        sortedKeyArray = this.nodeMap.keySet();
        return this.nodeMap.get(sortedKeyArray[i]);
}

/********************GETTING BOOTSTRAP***********************************/

function getBootStrap(){
        if(chord.nodeMap.empty())
                        /*If Chord is empty then return null as bootstrap*/
                return null;

        else{
                var rand = getRandomNumber(0,chord.nodeMap.length());
        /*If Chord is not empty then return any random node as bootstrap*/
                return chord.nodeMap.map[rand][1];
        }
}

function getRandomNumber(min, max) {
   return parseInt(Math.random() * (max - min) + min);
}
```

```
/********************GETTING BOOTSTRAP***********************************/

function getBootStrap(){
        if(chord.nodeMap.empty())
                        /*If Chord is empty then return null as bootstrap*/
                return null;

        else{
                var rand = getRandomNumber(0,chord.nodeMap.length());
        /*If Chord is not empty then return any random node as bootstrap*/
                return chord.nodeMap.map[rand][1];
        }
}

function getRandomNumber(min, max) {
   return parseInt(Math.random() * (max - min) + min);
}
```

```
/********************PRINT ALL NODES IN SORTED ORDER****************************/
Chord.prototype.printNodes = function(){
        for (var i = 0; i < this.nodeMap.length(); i++) {
                console.log(this.getSortedNode(i).printNodeInfo());
        }
}

/**************************CHORD ENDS***************************************/
```

## CHORDNODE.JS

```
/****************************NODE CONSTRUTOR*******************************/
function ChordNode(nodeId,nodeKey){
        this.nodeId = nodeId;
        /*Node IP:port or its id*/
        this.nodeKey = nodeKey;
        /*Contains hashed key of node*/
        this.fingerTable = FingerTable(this);                        /*Finger table of node*/
        this.predecessor = null;
        /*predecessor of node*/
        this.successor   = this;
        /*successor of node*/
        this.dataList = [];
        /*Data stored inside node*/
}


/****************************FIND SUCCESSOR******************************/
ChordNode.prototype.findSuccessor = function(key){
        if(this == this.successor)
                return this;

        if(isBetween(key,this.nodeKey,this.successor.nodeKey) || key == this.successor.nodeKey)
                return this.successor;

        else{
                var node = this.closestPrecedingNode(key);
                if(node == this)
                        return this.successor.findSuccessor(key);
                else
                        return node.findSuccessor(key);
        }
}

/********************CLOSEST PRECEDING NODE*******************************/
ChordNode.prototype.closestPrecedingNode = function(key){
        for (var i = KEY_LENGTH-1; i>=0; i--) {
                var finger = this.fingerTable[i];
                var fingerKey = finger.node.nodeKey;
                if(isBetween(fingerKey,this.nodeKey,key))
                        return finger.node;
        }
        return this;
}

/****************************JOIN OPERATION******************************/
ChordNode.prototype.join = function(node){
        this.predecessor = null;
        this.successor = node.findSuccessor(this.nodeKey);
}

/****************************IS BETWEEN OPERATION******************************/
function isBetween(key,fromKey,toKey){
        if(fromKey > toKey){
                /*When origin is coming between from and to key*/
            if(key>fromKey || key<toKey)
                return true;
        }
        else{
            if(key>fromKey && key<toKey)
                return true;
        }
        return false;
}

/***************************STABILIZE******************************/
/*  Verifies the successor, and tells the successor about this node.
        Should be called periodically.*/

ChordNode.prototype.stabilize = function(){
        var node = this.successor.predecessor;
        if(node != null){
            var key = node.nodeKey;
            if((this == this.successor) || isBetween(key,this.nodeKey,this.successor.nodeKey))
                this.successor = node;
        }
        this.successor.notifyPredecessor(this);
}

/**************************NOTIFY PREDECESSOR*****************************/
/* Notifies the predecessor about the joining of new node*/

ChordNode.prototype.notifyPredecessor = function(node){
        var key = node.nodeKey;
        if(this.predecessor == null || isBetween(key,this.predecessor.nodeKey, this.nodeKey))
            this.predecessor = node;
}

// /************************FIX FINGERS******************************/
// /* Refreshes finger table entries.*/

ChordNode.prototype.fixFingers = function(){
        for (var i=0; i < KEY_LENGTH; i++) {
            var key = this.fingerTable[i].start;
            this.fingerTable[i].node = this.findSuccessor(key);
        }
}

/**************************STORES THE DATA OBJECT*****************************/

ChordNode.prototype.storeData = function(data){
        this.dataList.push([data.dataId,data.dataValue]);
}

/**************************GET DATA VALUE*****************************/

ChordNode.prototype.getData = function(dataId){
        for (var i = 0; i < this.dataList.length; i++) {
            if(dataId == this.dataList[i][0])
```

```
                                        return this.dataList[i][1];
                }
                return null;
}

/**************************LOOK UP FUNCTIONS****************************/

ChordNode.prototype.lookUp = function(dataId){
                var dataKey = getDataKey(dataId);
                var node = this.findSuccessor(dataKey);
                var data = node.getData(dataId);
                if(data != null)
                                return data;
                else
                                return node.successor.lookUp(dataId);

}

/***************************NODE DISPLAY FUNCTIONS*****************************/
/* Gives information about the node */

ChordNode.prototype.nodeInfo = function(){
                return "NODE ID= " + this.nodeId + " NodeKey := "+this.nodeKey;
}

/*Prints the finger table of the node*/

ChordNode.prototype.printNodeInfo = function(){
                console.log("=====================================================");
                console.log("*********NODE : ID=> "+this.nodeId + "   KEY=> "+this.nodeKey +"**************");
                console.log("-----------------------------------------------------");

                if(this.successor != null) console.log("successor: " + this.successor.nodeInfo());
                else console.log("successor: null");

                if(this.predecessor != null) console.log("Predecessor: " + this.predecessor.nodeInfo());
                else console.log("Predecessor: null");

                console.log("-----------------------------------------------------");
                for (var i = 0; i < KEY_LENGTH; i++) {
                                var finger = this.fingerTable[i];
                                console.log(finger.start + "    " + finger.node.nodeId);
                }
                console.log("=====================================================");
}

function d2h(d) {return (+d).toString(16);}                /*Converts decimal=>hexadecimal*/
function h2d(h) {return parseInt(h,16);}                               /*Converts hexadecimal=>decimal*/

/***************************CHROD NODE ENDS*****************************/
```

## FINGERTABLE.JS

```
/*********************INTIALIZES FINGER TABLE*****************************/
function FingerTable(node){
var fingers = [];

for (var i = 0; i < KEY_LENGTH; i++) {
var start = startKey(node.nodeKey,i);
fingers[i] = new Finger(start,node);
}
return fingers;
}

/************************FINGER ENTRY************************/
function Finger(start,node){
this.start = start;
this.node = node;
}

/*********************START KEY GENERATOR*****************************/
function startKey(key,index){
var start_key = (key + Math.pow(2,index)) % MOD;                /*finger[k].start = (n + 2^(i-1)) mod 2^m*/
return start_key;
}

/*********************FINGER TABLE ENDS*****************************/
```

## 5.2 FIREFOX ADDON IMPLEMENTATION

### BROWSER.XUL

```
<?xml version="1.0"?>

<?xml-stylesheet href="chrome://helloworld/content/browser.css" type="text/css" ?>
                    <!-- including css file for xul -->
<!DOCTYPE overlay SYSTEM "chrome://helloworld/locale/browser.dtd">
                                                    <!-- Including dtd file -->

<overlay id="sample" xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">


          <stringbundleset id="stringbundleset">
          <stringbundle id="helloWorldBundle" src ="chrome://helloworld/locale/browser.properties" />
          </stringbundleset>

          <!-- Including the js file -->
          <script type="application/javascript" src="chrome://helloworld/content/browser.js"/>

          <!-- oncommand : activates when user hits button -->
          <hbox id="zotero-items-toolbar">
                    <toolbarbutton   id="network_search_button"
                                     tooltiptext="Network Search"
                                     style="list-style-image: url('chrome://helloworld/content/icon.png')"
                                     insertafter="zotero-tb-advanced-search"
                                     oncommand="HelloWorld.onCommand(event)"/>
          </hbox>

          <menupopup id="zotero-collectionmenu">
                    <menuitem label="Share on Network" oncommand="HelloWorld.onCommand(event)" />
          </menupopup>

</overlay>
```

### BROWSER.JS

```
let HelloWorld = {
          onCommand: function(event){
                    let stringBundle = document.getElementById('helloWorldBundle');
                    let hours = new Date().getHours();
                    let greeting = (hours < 12) ? stringBundle.getString("morningGreeting")
                                                                                            :
stringBundle.getString("eveningGreeting");
                    let message = stringBundle.getFormattedString('helloWorld',[greeting]);
                    dump(message + '\n');                              //Dumps the message to the console

                    window.open("chrome://helloworld/content/hello.xul", "", "chrome");
          }
};
```

## NETWORKSEARCH.XUL

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/global.css"  type="text/css"?>
<?xml-stylesheet href="chrome://helloworld/content/netSearch.css"  type="text/css"?>
<!DOCTYPE window SYSTEM "chrome://helloworld/locale/hello.dtd">

<window xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
                xmlns:html="http://www.w3.org/1999/xhtml"
    title="&title.label;">


<hbox align="center">
 <html:div id="actions">
    Your ID is <html:span id="pid"></html:span><html:br/>
    Connect to a peer:
    <html:input type="text" id="rid" placeholder="Someone else's id"/>
    <html:input class="button" type="button" value="Connect" id="connect" onclick="clickConnect();"/>
    <html:br/>
    <html:br/>
    <html:form id="send">
      <html:input type="text" id="text" placeholder="Enter message"/>
      <html:input class="button" type="submit" value="Send to selected peers"/>
    </html:form>
    <html:button id="close">Close selected connections</html:button>
  </html:div>

  <html:br/>
  <html:div id="wrap">
    <html:div id="connections">
        <html:span class="filler" id="fill" >You have not yet made any connections.</html:span>
        Hello
    </html:div>
    <html:div class="clear"></html:div>
        </html:div>

  <html:br/>
  <html:div id="box" style="background: #fff; font-size: 18px;padding:40px 30px; text-align: center;">
                Drag file here to send to active connections.
  </html:div>

        <html:br/>
  <html:div class="warning browser">
   <html:div class="log" style="color:#FF7500;text-
shadow:none;padding:15px;background:#eee"><html:strong>Connection status</html:strong>:<html:br/>
   </html:div>
  </html:div>

</hbox>

<script type="application/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/1.8/jquery.min.js"/>
<script type="application/javascript" src="http://cdn.peerjs.com/0.3/peer.js"/>
<script type="application/javascript" src="chrome://helloworld/content/netSearch.js"/>

</window>
```