Project Definition Document

PiTorrent

A web frontend for rTorrent, designed to run on a Raspberry Pi

Student	[Sean Heffernan]
ID	[0966654]
Supervisor	[Des Chambers]

Table of Contents

1 Introduction

The aim of this project is to build a system that will help end users to use their Raspberry Pi's as torrent sharing devices. I aim to build on the existing torrent client – rTorrent, and extend it so that it may be accessed through a web interface. The end goal is that users who have a Raspberry Pi at home, can log into their Pi through a web interface, and add/remove/monitor their torrent downloads.

1.1 Product Overview

The product is essentially a front end for rTorrent, that is aimed to be run on a Raspberry Pi, but should run on any Unix-based Operating System (OS).

RTorrent is a torrent client that runs over a Command Line Interface (CLI). Because of this, it has a steep learning curve as users have to learn a series of keyboard macros to use the software.

My product will create a Graphic User Interface (GUI) for rTorrent, which will allow users to combine rTorrent's good points (low footprint, stability) with the ease of use of a GUI.

Futhermore, the GUI will be web based. This is advantageous for the fact that it will enable users to access their torrents remotely. This plays to the Raspberry Pi's strengths, where they are suited to running headless (with no peripheral devices attached).

What sets my product apart from existing solutions is the fact that it should just work out of the box. Previous solutions rely on setting up a PHP web server with custom Simple Common Gateway Interface (SCGI) plugins that enable the GUI and rTorrent to communicate. My solution is built on Unix sockets, and does not need any of these surplus technologies.

1.2 Definitions, Acronyms and Abbreviations

OS	Operating System
CLI	Command Line Interface
GUI	Graphic User Interface
SCGI	Simple Common Gateway Interface
HTTP	Hyper Text Transmission Protocol
REST	Representational State Transfer
API	Application Programming Interface
RPC	Remote Procedure Call
XML	EXtensible Markup Language
I/O	Input/Output
MVC	Model-View-Controller

2 Requirments

2.1 Functional Requirments

The product has the following functional requirements:

1. It should allow users to administer their torrents through a web interface
2. The system should require users to enter a password in order to gain access
3. The system should allow users to browse and download their torrent downloads through HTTP requests made through the system.

2.2 Use Cases for Functional Requirments

Use Case 1- A user wishes to add a torrent to his Raspberry Pi and delete it once its finished downloading.

Flow of events:

1. The user accesses the system by typing in the systems address on a web browser.
2. The user logs in to the system (use case number 2).
3. The user uploads a torrent file to the server.
4. The server starts the download.
5. The torrents progress is shown via the GUI.
6. When the torrent is finished, the user clicks a button to remove the torrent file (the actual download remains on the server)

Use Case 2 - A user wishes to access the system.

Flow of events:

1. The user accesses the system by typing in the systems address on a web browser.
2. If the system finds a valid session on the users web-browser, the user will be logged in automatically.
3. If the user does not have a valid session, they will be prompted for a username and password.
4. If the username and password are correct the user will be logged in.

Use Case 3 - A user is on a firewalled network where torrent traffic is blocked. The user wishes to download a torrent to their Raspberry Pi (which is not on the firewalled network), and when the download is complete, download the torrent from the Raspberry Pi to their local machine over HTTP, thus bypassing the firewall.

Flow of events:

1. The user accesses the system by typing in the systems address on a web browser.
2. The user logs in to the system (use case number 2).
3. The user uploads a torrent file to the server.
4. The server starts the download.
5. The torrents progress is shown via the GUI.
6. When the torrent is finished, the user clicks a button to download the torrent data.
7. The torrent is transferred from the Raspberry Pi to the client over a HTTP download.

2.3 Non-functional Requirments

The following non-functional requirements have been identified:

1. The server should communicate to rTorrent via Unix domain sockets.
2. The system should maximise the amount of computation done on the client, offloading the “work” done on the Raspberry Pi.
3. Following on from previous requirement, the server should make extensive use of Representational State Transfer (REST) web services. This means that the server should provide a range of Application Programming Interfaces (API's) that the client can connect to, in order to recieve data.
4. The system should adhere to responsive web design principals - the GUI should look good regardless of the device used to access it.

2.4 General Constraints

The system is built for a Raspberry Pi, which uses an ARM processor. Any software used in making this system must be capable of running on the ARM architecture.

As you can imagine, the Pi not as powerful as an average household computer. Therefore, any technology used in this system must be capable of running smoothly on the Pi. Also, the code used to make this product must be well designed and as efficient as possible.

RTorrent has an Remote Procedure Call (RPC) interface. This allows us call rTorrent functions by sending EXtensible Markup Language (XML) messages to this interface, and recieve XML responses back. The system should communicate with rTorrent's RPC interface through Unix domain sockets, as opposed to the alternative option, communicating through HTTP and Simple Common Gateway Interface (SCGI) requests.

3 Technologies and Frameworks

3.1 Raspberry Pi

For those of you not familiar with the Raspberry Pi, it is a credit-card sized computer that has gained huge popularity since its release in February 2012. This single-board computer is developed by the non-profit group, the Raspberry Pi Foundation. The purpose of the Raspberry Pi is to promote the teaching of basic computer science in schools. But because of its form, power, and price-tag (35 USD for the advanced version), it has gained something of a cult following among gadget enthusiasts around the world.

The Pi is well suited to acting as a file sharing server for a few reasons:

1. It is small – its the size of a credit-card.
2. It uses a small amount of power (3.5W)
3. It is cheap to buy - $35
4. It has USB ports that allows us to connect external hard-drives to boost the storage space.

3.2 Raspbian

Raspbian is a Linux distribution that was built especially for the Raspberry Pi. As you may have guessed by looking at the name, it was created as a port of Debian Linux OS, a very popular Linux distribution, upon which Ubuntu is built. I have chosen to build my project for Raspbian, as it is probably the most popular distribution for the Raspberry Pi, and has support for a great selection of software that I can use in my project.

3.3 RTorrent

RTorrent is a CLI based torrent client. Its written in C++ and has no GUI. Because of this, it is fast and has a low footprint – making it the ideal torrent client. It also uses a text file to store any configurable options (download directory, etc). This text file can be easily manipulated from a web interface.

RTorrent is a very popular torrent client, and is included in the default repositories for many of the big Linux operating systems.

My project exploits rTorrent's RPC interface help to extend its capability.

3.4 Node.js

Node.js the platform that I have chosen to run on the Raspberry Pi which will communicate with rTorrent and serve HTTP requests. Node.js is built on top of Google's v8 JavaScript engine, and has gained a lot of popularity in recent times, with industry giants like Microsoft and Linkedin using it for production sites.

In simple terms, Node.js allows us to write a web-server in JavaScript. This in itself is cool, but the reason it has gained such a following is because it enables us to build highly scalable network applications. Node uses an “event driven, non-blocking Input/Output (I/O) model that makes it lightweight and efficient”.

Node runs in a single thread, but any I/O is done in event loops. These event loops make use of JavaScript's callback functionality and allows the I/O to be non-blocking.

What makes Node suitable for my project is the fact that its lightweight and supports Unix socket connections. It's also fast to develop in Node as you don't have to worry about concurrency issues, and also for the fact that Node has a fantastic set of packages that we can make use of, e.g. XML parser, authentication system.

Here is all the code required to create a simple “Hello World” web-server:

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
}).listen(1337, "127.0.0.1");
console.log('Server running at http://127.0.0.1:1337/');
```

To run this program, simply save the code above in a file, and run the command : **node <filename>**

3.5 Jade

Jade is a Node.js templating engine. It is a language that compiles to HTML. It includes some nice features such as 'includes', where you can include one Jade file in another. This allows us to separate different GUI components into different files, and makes the presentation layer more modular. Using Jade should help us to write HTML much quicker and we will always end up with valid HTML documents when using Jade.

Here is an example Jade document that compiles to fully valid HTML:

```
doctype 5
html(lang="en")
  head
    title= pageTitle
    script(type='text/javascript').
      if (foo) {
         bar(1 + 5)
      }
  body
    h1 Jade - node template engine
    #container.col
      if youAreUsingJade
        p You are amazing
      else
        p Get on it!
      p.
        Jade is a terse and simple
        templating language with a
        strong focus on performance
        and powerful features.
```

3.6 RESTful Services

REST allows us to define API's where we can get, modify or delete resources all over HTTP. In my project for example, I could define a resource at '/torrents'. Depending on the HTTP request type I access this address at (GET, POST, DELETE, etc), I can get, delete or modify torrents. What's nice about using RESTful services to transfer information is that I can have JavaScript running on a clients browser that will issue GET requests to '/torrents' every two seconds and update the browser when it gets a response. The entire page will not be refreshed, only the torrent data (torrent upload speed, download speed, etc).

3.7 AngularJS

AngularJS is a JavaScript framework that is designed to run on the client side. Maintained by Google, it aims to provide a Model-View-Controller (MVC) paradigm to client side JavaScript programming. AngularJS allows us to augment and extend existing HTML with further functionality such as two-way data binding that automatically synchronises the model and view.

What this means is that if we show a variable in our view (e.g. a torrent's upload speed) and we then update this variable (via a JavaScript GET request to the server's REST API), the view will automatically be updated with the new upload speed. By using AngularJS we can drastically cut down the amount of boilerplate JavaScript code we have to write.

Here is an example of AngularJS and Jade working together. In this example, we have a JSON object with an array of torrents (torrentRestults.torrents). We use the Angular “ng-repeat” directive to loop through all the torrents in this array that are seeding (uploading), we then order the torrents by name, and limit the number of torrents shown to 5.

We can then display torrent information by using the {{ }} notation.

```
ul.project
  li(ng-repeat="torrent in torrentResults.torrents | filter:seeding | orderBy:'name' | limitTo:5")
    strong {{torrent.name}}
    p.p-meta
      span Downloaded : {{torrent.downloaded}}
```

AngularJS has a lot of functionality, I plan to use it extensively to build up and update the clients view.

3.8 Bootstrap

Creating an attractive HTML web-app is no mean feat. After all, I am a programmer, not a designer. For this reason I have chosen to use Bootstrap in my project.

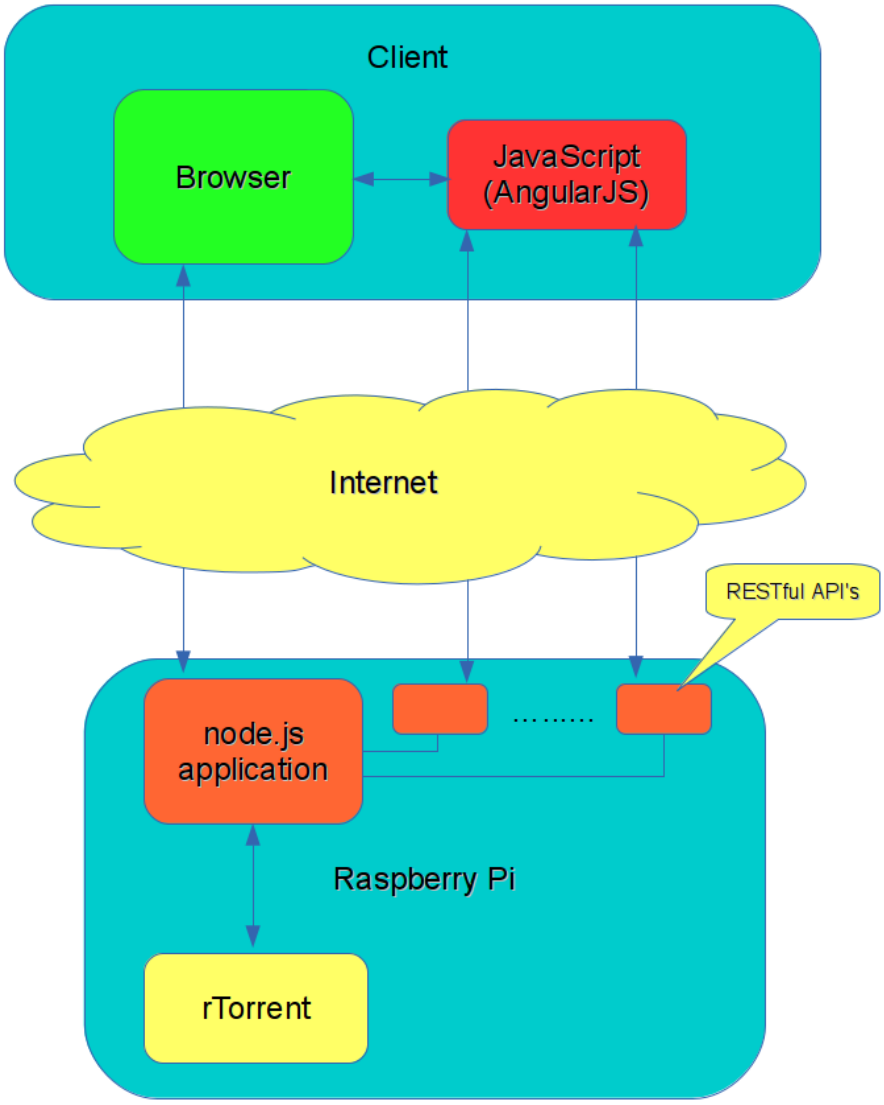
Bootstrap is a framework that hopes to streamline HTML development by providing users with a range of styled HTML components such as buttons, dropdown menus, navbars, etc. It also incorporates a grid system and responsive design principals.

We can simply include the Bootstap JavaScript and CSS files in our project and drop the code for these components into a HTML page.

4 System Architecture

Here is a high-level overview of the project's components:

- The client will connect to the Raspberry Pi over HTTP through a network connection.
- The node.js application will hear incoming requests and serve web pages to the client.
- At regular intervals, client-side JavaScript will send requests to the server's RESTful API's to get up-to-date torrent information (upload speed, download speed, etc.).
- These requests will be handled by the node.js application, which will then connect to the rTorrent process via a Unix socket and get the specified information.
- The node application will then convert the rTorrent XML response into a human readable JSON object and send this object back to the client via HTTP.
- The client side JavaScript will update the browser page with the newest torrent information.



5 Work Breakdown Structure

5.1 Research

- **rTorrent**

  I will have to investigate Investigate what functions we can call through rTorrent's RPC interface. There is no official rTorrent documentation listing what function calls we can make. Luckily, rTorrent is open-source so I can examine the code itself.

- **Study documentation for the various technologies I plan to use**

  I have not had much exposure to coding in many of the technologies I plan to use. I will have to investigate these technologies in the hope of finding “best practices” that should help me write better, more maintainable code.

- **Investigate responsive design principals**

  I will have to research solutions for displaying a web-app across multiple devices. Bootstrap claims to help in this regard, but since I've never used it for this purpose before, I feel I need to give it further research.

5.2 Design

- **The structure of the Node.js application**

  The Node.js application will be the largest part of the product. It has to provide authentication, serve HTML pages, provide RESTful API's to the client, and handle communication with rTorrent.

- **What RESTful API's the Node.js application should provide**

  RESTful API's will provide the means for the client and server to transfer information. I will need to design a number of these well defined API's that the client can call in order to get information.

- **The structure of the client-side JavaScript**

  I plan to farm out as much data processing as I can to the client in order to keep the Raspberry Pi as unburdened as possible. This means that there could be a large amount of client-side JavaScript required – which will need to be designed.

- **How should I present data to the end-users**

  The way in which data is presented to users is pivitol to the success of the product. Users should be able to easily process what rTorrent is doing at a given moment in time. In order to achieve this, I will need to design an attractive and informative GUI.

5.3 Development

- **Node.js application**
- **Client-side JavaScript**
- **GUI HTML**
- **Testing**

  I decided to consider 'testing' as a development activity as I plan on taking an agile approach to development.

6 Bibliography

- Raspberry Pi - http://www.raspberrypi.org/
- Raspbian - http://www.raspbian.org/
- RTorrent - http://libtorrent.rakshasa.no/
- Node.js - http://nodejs.org/
- Jade - http://jade-lang.com/
- AngularJS - http://angularjs.org/
- Bootstrap - http://getbootstrap.com/