National University of Ireland, Galway

CS & IT Final Year Project Thesis

PiTorrent

Author:
Sean Heffernan

Supervisor:
Dr. Des Chambers

A thesis submitted in fulfilment of the requirements
for the degree of Computer Science & Inforamtion Technology

March 2014

Declaration of Authorship

I, Sean HEFFERNAN, declare that this thesis titled, 'PiTorrent' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

"Anyone who has lost track of time when using a computer knows the propensity to dream, the urge to make dreams come true and the tendency to miss lunch."

Tim Berners-Lee

Acknowledgements

Acknowledgements go here...

Contents

Abbreviations

IDE	**I**ntegrated **D**evelopment **E**nvironment
HTTP	**H**yper**T**ext **T**ransfer **P**rotocol
P2P	**P**eer-**T**o-**P**eer
GPU	**G**raphics **P**rocessing **U**nit
CLI	**C**ommand **L**ine **I**nterface
SSH	**S**ecure **S**hell
RPC	**R**emote **P**rocedure **C**all
SCGI	**S**imple **C**ommon **G**ateway **I**nterface

Chapter 1

Introduction

1.1 Welcome

Welcome to my thesis for my project PiTorrent. PiTorrent is a web-based torrent management system that is aimed to be run on a Raspberry Pi. The system should essentially allow users to easily turn their Raspberry Pi into a file sharing system.

With this project I aim to fill a gap in the market for such a product. A small number of projects that claim to give a similar service already exist, but they are difficult to get working and involve installing and configuring a number of various programs. I will talk more about rival products later, I just want to say now that my project will solve many of the problems with the current paradigm.

So with that said, lets start at the beginning. What are torrents and why should we care about them?

1.2 Torrents and the BitTorrent Protocol

Traditionally, if we download a file from the internet, we open a connection with a web-server and download the file from this web-server. Just two parties are involved in the transaction, us (the client) and the web-server. This model is good and works well, but has some possible shortcomings. For example, say I wish to download the Eclipse Integrated Development Environment (IDE), which is located on a web-server in America. My friend, who I'll call Chris, also wishes

to download Eclipse. We are both located in Galway, Ireland. In order to download Eclipse, we both have to open connections to the web-server in America and transfer the same data across the Atlantic.

Now lets just think about this. When Chris and I are downloading the files from the remote web-server over Hypertext Transfer Protocol (HTTP), we are getting the web-server to send the same data across the Atlantic twice. Internet traffic sent across the Atlantic tends to be slower due to network congestion. We are also putting more strain that is necessary on the web-server, as both Chris and I are downloading the file in its entirety twice.

But what if Chris and I could somehow share our downloads, if we could each download a different half of the file from the web-server in America and share the data we have downloaded amongst ourselves. The advantages of this are clear to see:

- We only need to put half the "strain" on the web-server
- We can get increased download speeds as we are downloading off multiple sources
- We can download from a more "local" source - which can also help download speeds

Because of the advantages of a system like this, the BitTorrent protocol was created.

1.2.1 The BitTorrent Protocol

Bram Cohen, an American computer programmer designed the BitTorrent protocol in 2001. He released his first implementation of it in July 2001 [**?**]. Bram's vision was to enable simple computers, such as home PC's to share file amongst themselves. The protocol works as follows:

- A user who wishes to share data creates a torrent descriptor file
  - This is a .torrent file that defines segments in the data called "chunks"
  - A cryptographic hash is generated for each chunk, and the hashes are stored in this descriptor file
  - These descriptor files are quite small, typically a few kilobytes in size
- The user distributes the descriptor file by any means it wants to other users who wish to download the data

- The user then makes the data it wishes to share available through a BitTorrent node
- Other users can connect to this node, via links in the torrent descriptor, and download the data. They download data in chunks
- Once users start to download chunks, they can share the chunks they have downloaded with other users - reducing the load on the original uploader
- Any chunk downloaded can be verified by generating a hash for the chunk and checking this hash against the hash in the torrent descriptor file

The protocol is implemented by programs called BitTorrent clients. There are multiple clients available to download; uTorrent, rTorrent and Vuze are a few popular options that come to mind.

BitTorrent has become a well established protocol over the years, with many websites that serve large files adding BitTorrent download links.

1.3 The problems with running a BitTorrent server

BitTorrent is a peer-to-peer (P2P) protocol where the overall "health" of a torrent relies on users keeping their torrent client open and uploading the torrent data back to other users. If all users stopped their torrent client after they downloaded the torrent, it would not take long until there were no "seeders" on the network, i.e. users who have the all the data and are uploading it back to the community. Some BitTorrent communities exist where users share files with each other exclusively, and each user is asked to maintain a 1:1 ratio - meaning they upload as much data to their peers as they download from their peers. Failing to keep this ratio will result in the user being removed from the community.

Because of this, serious BitTorrent users usually leave their BitTorrent clients running for extended periods of time - until they upload as much data as they have downloaded. This in itself may prove problematic, as it can be inconvenient to leave your laptop or PC running unattended for hours on end. Therefore enthusiasts usually resort to resurrecting old PC's or buying cheap PC's that they can use solely as BitTorrent servers that they run 24/7.

These servers are usually run headless (without a monitor, keyboard and mouse) and are administered via a network connection. As a result of this, the torrent clients that these servers

use need to be web-based, where users can log into their client over a network connection and monitor their torrents. There has been some advancements in this area in recent years with many good web-based torrent clients coming into existence.

1.4 The Raspberry Pi

The Raspberry Pi is a pocket sized computer that is developed by the non-profit group, the Raspberry Pi Foundation. The Pi was initially released in February 2012 as a cheap computer to promote the teaching of computer science to kids in schools. The Pi was priced at 35 USD and was capable of running a full Linux stack, as well as having a Graphics Processing Unit (GPU) of playing full HD 1080p video. The project generated huge hype and interest due to the "bang-for-buck" of the Pi's specifications. Any self respecting gadget enthusiast put themselves on a waiting list for a Pi.

Due to the Pi's price tag, form factor and low power consumption they make an excellent torrent server. Raspbian, a port of the popular Linux distribution was created especially for the Raspberry Pi. Raspbian comes with over 35,000 packages especially compiled and optimised for the ARM architecture of the Raspberry Pi. As luck would have it, Raspbian comes with some torrent client packages that can easily be installed. However, many of these clients are quite CPU intensive and are reported to make the Pi sluggish. The torrent client that has the lightest footprint, and therefore the best suited for the Pi's limited resources is called rTorrent.

1.5 rTorrent

rTorrent is a text based BitTorrent client that runs over a Command Line Interface (CLI). It is written in C++ and was developed with a focus on high performance and low memory usage. It has become a very popular torrent client due to its stability and efficiency. But it is not without its issues. As rTorrent is a CLI based program, it has a learning curve as users need to learn keyboard macros such as pressing CTRL+s to start a torrent and CTRL+d to stop a torrent. rTorrent also is not web-based, although users can log into a server via SSH (Secure Shell) to administer their torrents over the command line.

In over to overcome these issues, numerous attempts were made by 3rd party individuals to code a web interface to rTorrent by calling functions on rTorrent's exposed Remote Procedure Call

(RPC) interface. Some of these attempts were quite good, with good products like ruTorrent and TorrentFlux being moderately successful.

The problem with these solutions is that they are by no means simple to get working. They rely on a number of different technologies that need to be installed and configured to work together. Technologies like PHP, Apache, and Simple Common Gateway Interface (SCGI) plugins for Apache to enable PHP and rTorrent to communicate with RPC messages. Remember that in many cases, the server on which you are wishing to install these products on are headless - so this installing and configuring needs to be done entirely over the command line, a daunting task to the uninitiated in Linux admin.

So this is where I come in. My idea is create a web interface for rTorrent that will do away with the hardship of setting up such a system. I propose to use UNIX sockets to communicate with rTorrent's RPC interface, which has not been done before. With this approach I can remove the necessity to install a web server such as Apache, and configuring it with SCGI and PHP plugins. My system should be a "one click install" product that also incorporates modern development requirements like responsive design - something that rival products lack.

Chapter 2

Technologies

2.1 Foreword

One of the intentions from the outset of this project was to embrace new technologies. This does not mean that I jumped on the bandwagon of every up-and-coming hipster product coming straight out of Silicon Valley. I carefully and thoroughly investigated some of the leading "newer" technologies that I felt had genuine potential to become proven mainstream solutions. I discarded some of these technologies, others I chose to use at the core of my project.

2.2 Node.js

My project has two main sections, the front-end (which users will see on their web browser), and the back-end (which acts as a web-server on the Raspberry Pi). The back-end acts as the middleman between users and rTorrent. If a user clicks a button to stop a torrent in the front-end, the front-end sends a HTTP message across the network to the back-end, the back-end then builds a XML-RPC message to stop the torrent, and sends the XML-RPC message to rTorrent via a UNIX socket. A rTorrent "stop" function is called, and the response is sent back to the back-end, which in turn builds a HTTP response, and sends it to the front-end over the network. Once the front-end gets the response, it update the view to show the torrent has stopped.

The back-end is the core of my project, and choosing a technology to use as the back-end was the biggest decision I had to make while creating PiTorrent. The technology should be:

- ARM compatible - must run on an ARM CPU
- Lightweight - must not hog the Pi's limited resources
- Easy to install - preferably be in Raspbian's software repositories
- Easy to use - possible to get a prototype up and running quickly

After some consideration I decided to use Node.js as my back-end technology. Node.js is a platform that allows us to use JavaScript as a back-end programming language. It is geared at building fast and scalable net applications. It comes pre-compiled for ARM architecture, so it will run on the Raspberry Pi without any problems. Node's main selling point, and the reason I choose to use it in my project, is it's threading model. Node.js makes use of JavaScript's callback functionality to enable an event driven, non-blocking I/O model.

One of the most difficult issues to overcome in building server-side web application is threading. For example, with Java you would use an application server like WebSphere or GlassFish to handle thread generation and management for you. Even when using these middleware applications, dealing with threads still proves difficult. This is where Node.js I/O model comes into play. With Node, there is only one main execution thread. All I/O calls are non-blocking and when a result is returned, an event is generated and a callback is executed.

Here is an example of some JavaScript that runs under a Node.js environment. Node comes with a 'net' package that allows us to make use of sockets in our code. This example shows all the code required to connect to a UNIX socket (located at '/tmp/rpc.socket') and write and read data from the socket. Notice the layout of the code - when we try to connect to the socket (line 2), we do not wait for the connection to be established, the thread of execution moves onto the next instruction. In this case, that next instruction is line 11 - printing "Coffee time!" to the screen. Only when the connection has been established, a "connect" event is generated and the thread of execution runs the unnamed function which starts on line 3.

```
1  var socket = new net.Socket().connect("/tmp/rpc.socket");
2  socket.on("connect", function() {
3    socket.write("Hello!");
4    socket.on("data", function(data) {
5      // Do something with the incoming data
6    });
7    socket.on("error", function (err) {
```

```
8       console.log(err);
9     });
10  }
11  console.log("Coffee time!");
```

LISTING 2.1: Node.js and event driven I/O

With this threading model, we don't have to worry about concurrency issues. This will enable us to build quality applications in a much quicker time-frame compared to other platforms.

Another selling point of Node.js is the fact that it is very lightweight. The current Node binary for ARM is 5MB in size. Compare this to Java 7 and GlassFish which is a whopping 140MB combined.

2.2.1 Express - a Web Application Framework for Node.js

Node.js by itself is quite low-level. It allows us to build web applications from the ground up, exactly the way we want them. However, the majority of Node users do not want to spend their time building authentication systems, or URL route handling systems, or other boilerplate code - they want to be writing business logic. This is where Express, a Node web application framework comes into play. Express takes care of all this boilerplate coding that every web project requires. By using the Express module in our Node applications we can use their URL route handling and other neat features like managing view engines, like Jade.

2.2.2 Jade - a Node.js Template Engine

Appendix A

Appendix Title Here

Write your Appendix content here.