Utilising peer-to-peer networking for data transfer over the browser

Dominic Rathbone

April 15, 2016

Contents

Chapter 1

Project Scope

1.1 Aims and Objectives

The ultimate objective of my project is to investigate the plausibility of data transfer over web browsers (in particular, with file transfer and media streaming) without the need for a centralised client-server architecture, instead opting for a peer-to-peer network architecture. In order to do so, I plan to:

- Research peer-to-peer networking architecture and topologies
- Research WebRTC
- Research signalling protocols
- Research media streaming compression & protocols
- Research client-side JavaScript frameworks
- Develop a session signalling server in Java with the WebSockets protocol
- Develop a WebRTC Application to transfer and stream uploaded files.
- Implement a structured peer to peer network between peers transferring/streaming a file using the web application.

1.2 Stakeholders

The stakeholders involved in my project will be myself, my supervisor, Stelios Kapetanakis and the user.

1.3 Methods of Communication

Stelios and I have set up a regular meeting once a week on Friday at 4pm to catch up. On top of this, we communicate regularly via email and Stelios has access to a project git repository on GitHub and a workflow board set up on my web server to track the progress of my project.

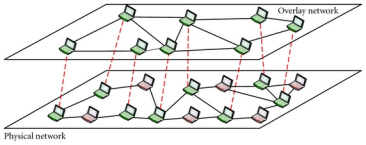
Chapter 2

Research

2.1 Comparing Network Architectures

Peer-to-peer networking is the distribution of resources and processing between the nodes of a network. These networks tend to take the form of an overlay network which is a topology describing a virtual network that sits on top of a physical network (e.g the internet) consisting of a subset of the nodes connected to the physical network.

Figure 2.1: An Unstructured Peer-To-Peer Overlay Network [1]



Although an overlay network does not need to manage the burden the physical network is responsible for, in order to discover and route new nodes, it needs to have some way of managing the subset of nodes using it. There are several different ways of defining how it does this:

- Unstructured peer-to-peer: Peer connections in the overlay are established randomly and do not adhere to a network topology. New peers copy the connections another peer has formed and develops it's own over time.[2]. These networks are easier to build than structured networks as routing is random and they also tend to fair better in periods when the rate of peers leaving and joining (churn rate) is high due to their non-deterministic nature. However, searching in these networks tends to be less efficient as they rely on flooding the network to find nodes that have the data they are searching for.

- Structured peer-to-peer: Peers in a network are organised in a specified network topology and generally use a distributed hash table (DHT) to link together. The DHT consists of many peers maintaining a partial hash table referencing the unique keys of other peers in the network, allowing it to route to them. This way, a peer can communicate with another by hopping messages through peers in their hash table. Generally, each peer's hash table consists of nodes getting exponentially further away from it in order to improve the efficiency of this process. [2].

- Hybrid peer-to-peer: A combination of peer-to-peer and client-server models, these networks use centralized servers in some way. This model tends to work better than both unstructured and structured as dependent on how it is implemented, features that function better in decentralized networks can be left to the peers in network whilst the server can handle the features it is better at. [3]

Additionally, there is also the client-server model. In a basic sense, this is when the machines processing data (servers) are distinct from the computers requesting this processing (client), although in actuality, the

boundaries between them vary dependent on how it is implemented. Whilst the shift of processing on to the service supplier means that the user of the application does not need a powerful machine to run it, it also means that the service supplier has the responsibility of maintaining and paying for these servers. Furthermore, although modern server architectures are designed in a distributed structure granting them to be more robust (as distribution allows for redundancy and load balancing), if these servers undergo too much load or in extreme cases, if the server farm completely crashes, the user would not be able to use the application as intended.

Using a peer to peer architecture avoids the issues associated with the client-server model as processing is distributed between the nodes making up the network, resulting in it being cheaper to maintain as there is less reliance on servers and more robust as the network scales as load increases. However, there are a number of considerations to be aware of in peer to peer networking, the largest of which being security as there is no centralized authority to manage access control to resources. Thus, a malicious client could potentially launch a number of attacks versus other nodes in the network such as Denial Of Service (DoS) where a malicious group of nodes floods a network with a massive amount of fake data, potentially crashing nodes within it or Man in the Middle (MitM) attacks where a node is able to intercept data flowing between two other nodes and spy on or poison this data[4]. Whilst it would be hard to completely avoid risks like this as it is an inherent flaw in peer-to-peer architecture, introducing the basic concept of trust into a peer-to-peer network can stop malicious peers from entering it in the first place. This will be implemented in my application by allowing users to enable authentication (via password) for the uploaded file potentially preventing malicious attackers from being able to download or stream it.

My application will most closely resemble a hybrid peer-to-peer network as even though the actual channel for transferring data between peers (WebRTC Data Channel) will not go through a server, it will use a signalling server to route peers together. To form a network of peers that go beyond the one-to-many peer connection established by WebRTC, the signalling server in conjunction with the client-side application will have to manage how these peers connect to each other. A way of doing this is to connect new peers to the second newest peer and so on, forming a linear chain of peers that can reroute if the neighbouring peer disconnected.

Figure 2.2: Linear network

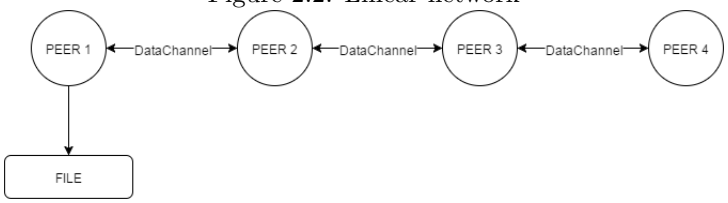
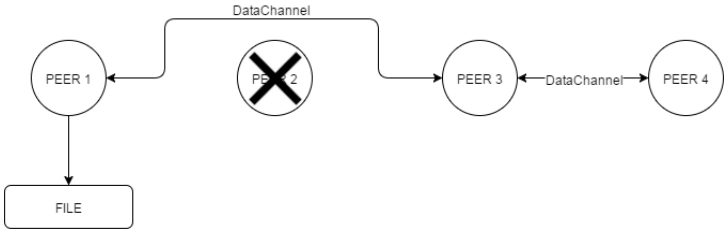


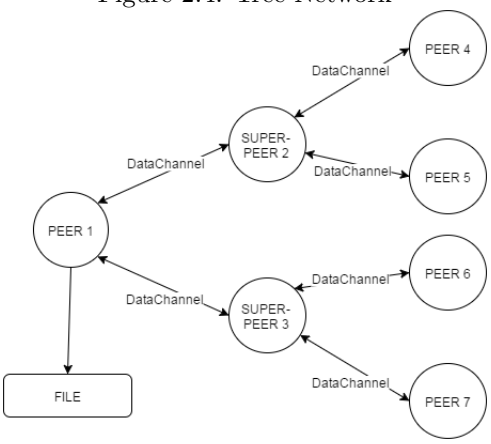
Figure 2.3: Peer disconnects from the network



However, this would prove inefficient if a peer in the chain malfunctioned or restricted bandwidth, potentially bottlenecking the rest of the peers behind it. As an alternative, peers could be routed based on a tree network topology [5] consisting of supernodes, "high capability" nodes that relay the transferred data to subnodes. This way load can be balanced across many nodes by distributing new nodes to another supernode or by electing further supernodes if it increases too much or if one disconnects. By exploiting heterogeneity between peers of the network, this topology can also be used to improve performance within the network by electing supernodes based on criteria such as location and, in our case, how much of the

data has already been transferred or streamed to them. By doing so, performance within a network can potentially be improved as subnodes can be connected to supernodes based on this criteria, e.g if a subnode is located in the same place as a supernode then use it.[6].

Figure 2.4: Tree Network



2.2 Current Solutions to Data Transfer

Currently, solutions such as DropBox and Google Drive focus on cloud storage as a service whilst providing the ability to transfer data as a secondary function of this service. Although this is an extremely valuable and convenient idea enabling consumers to back-up their data, not all want a third party service that stores the data they want to transfer as this practice does raise ethical issues, namely with the data's security, privacy and ownership once it has been uploaded. However as a consumer, this can be hard to avoid when the online storage and transfer of data are so closely intertwined as these services have a huge market share.

Once you have uploaded files to one of these services, there is a trust placed on the company to store the data securely. However you could argue that this trust is misplaced, in 2010, the "Cloud Security Alliance" released a regularly updated list of the top threats faced by cloud computing services submitting there are numerous security issues faced within the cloud computing industry with many of the issues on this list occurring due to malicious intentions. For example the 6th item on the list, "Malicious Insiders" describes a situation in which current or former employees abuse their authorizations within system to gain access to sensitive information [7]. This suggests that even though companies can protect their client's data to an extent, there are still issues the industry face that consumers may want to avoid.

Whilst the uploaded data can be encrypted in a way that means only clients of the service can decrypt it (through end-to-end encryption), many people feel insecure leaving traces of their private information on a remote server belonging to a company that is not necessarily always acting in their interest. In 2013, this concern was legitimized by Edward Snowden when he disclosed information about government surveillance programs such as PRISM that coerced firms such as Google, Microsoft and Yahoo to provide private consumer data from their services to the government.[8] More recently, this idea has been reinforced by legislation such as the "Investigatory Powers Bill" being introduced in the UK that could prohibit companies from using end-to-end encryption techniques allowing authorities to request decrypted client data [9]. Whilst we can assume they have done this with non-malicious intentions, by weakening the encryption practices of these companies, they will potentially weaken the defence against malicious attackers.

As an alternative to using cloud storage solutions for sharing files, there is also specific data transfer sites such as WeTransfer that allows you to send files up to 2gb via their site. This works by sending a link through email to the person you want to send the file to. Whilst it does store the uploaded files, they are only kept for 7 days in order to prevent unnecessary storage costs [10]. Although this is better for data privacy, WeTransfer use third party providers such as Amazon Web Services (AWS) to store this data using AWS S3 Storage [11] meaning there is still the potential concern of data privacy and security highlighted previously.

Peer-to-peer networks also been popular for file sharing since the late 1990s due to the rise of applications such as Napster which focused on peer to peer music sharing. Napster worked by using a central server to index files that each peer made available on their machine. Each peer would then be able to search the index for copies of songs and download them from each other. However, this service was eventually shut down as the company ran into legal problems with copyright infringement. Since then, protocols such as BitTorrent have improved on the idea of peer to peer sharing. The BitTorrent protocol works by peers creating ".torrent" descriptor files that describe the file's metadata, this is then shared however it is seen fit. Each peer wanting to download this file then picks up this ".torrent" file, opens it with a BitTorrent client and joins a "swarm" of peers, becoming one of the many "leechers" downloading this file. At the same time, this peer becomes one of the many "seeders" uploading this file as they download it so that other leechers can download it from them. The initial peer discovery is based on "trackers", servers hosting lists of seeders and leechers for the file. One of the issues with BitTorrent is that in order to share a file, you must go through the process of downloading a client, creating a ".torrent" file, adding it to a trackers list and sharing the ".torrent" with your peers and this can be seen as a arduous task for a basic consumer looking to share their files. Another problem is that if a tracker is compromised, there is potential for malicious attacks using the information stored on there.

As an alternative to these current solutions, the application I plan to develop has the aim of separating the online transfer of data from it's storage by connecting the peers involved in the transfer directly with each other. This circumvents the problems with cloud storage as the data being transferred does not pass through any servers as well as the problems with current peer to peer file sharing as it does not require a desktop client or torrent files. To allow my application to share files online, WebRTC will be used.

2.3 WebRTC

WebRTC (Real Time Communication) is an emerging web technology that enables browsers to communicate real time via a peer-to-peer connection, avoiding the need for a centralized server to transfer data between clients. This was first released by Google as an open source project in May 2011 [13] and was later drafted as an API definition by W3C remaining as a work in progress. [14]. WebRTC has yet to be fully implemented in every web browser but Chrome, Firefox and a WebRTC specific browser called Bowser do support it. Firefox (Nightly), in particular seems to be leading the way in this so the application will specifically be developed towards this browser[15]. The 3 main WebRTC APIs supported at this time are:

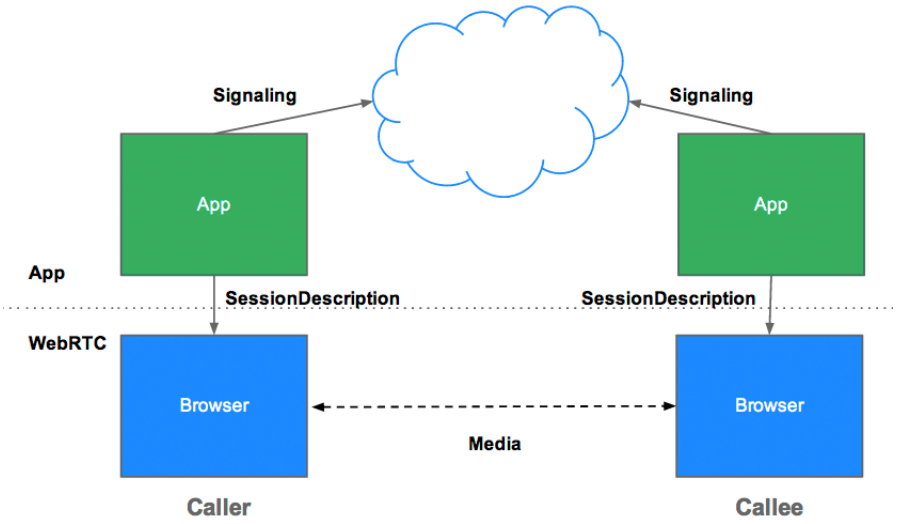
- RTCDataChannel: "The RTCDataChannel interface represents a bi-directional data channel between two peers of a connection." [16]
- RTCPeerConnection: "The RTCPeerConnection interface represents a WebRTC connection between the local computer and a remote peer. It is used to handle efficient streaming of data between the two peers." [16]
- getUserMedia: "Prompts the user for permission to use one video and/or one audio input device such as a camera or screensharing and/or a microphone. If the user provides permission, then the successCallback is invoked with the resulting MediaStream object as its argument. If the user denies permission or media is not available, then the errorCallback is called with PermissionDeniedError or NotFoundError respectively. Note that it is possible for neither completion callback to be called, as the user is not required to make a choice."[16]

In order to achieve it's aim, the application will utilize the RTCPeerConnection and RTCDataChannel APIs. The former to establish a peer connection between two clients and the latter to create a data channel over this peer connection to transfer data.

As mentioned in the Javascript Session Establishment Protocol (JSEP) [17], although WebRTC and the browser is used to transfer the data between two peers, it purposely does not handle signalling. This is a concept that came from telecommunications and utilised in VoIP and is the process of organising the communication between two clients, handling the exchange of metadata that creates and manages a session. The rationale behind WebRTC being signalling protocol-agnostic is that different applications will require

particular protocols in order to, for example, fit into previously existing architecture. In the case of my application, to handle signalling, Session Initiation Protocol (SIP) over WebSockets will be used.

Figure 2.5: WebRTC Architecture as defined by JSEP [17]



WebSockets is a protocol implemented by browsers and servers to provide bi-directional communication between them[18]. Once established, a session with a client communicating with the server is left open for the server to send messages to the client and vice versa, making it a good candidate for signalling with a WebRTC application which needs peers to be able to reliably send messages back and forth through the server. Over this WebSockets session, SIP messages will be sent. SIP is a communication protocol that does not specify how signals are transported but how these signals are defined. The transaction model defined by SIP is similar to HTTP with each SIP request being matched by a SIP response. An example of a SIP over WebSockets transaction:[19]

Initial handshake with WebSockets signalling server over HTTP
Alice -> proxy.example.com (TLS)

```
GET / HTTP/1.1
Host: proxy.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: https://www.example.com
Sec-WebSocket-Protocol: sip
Sec-WebSocket-Version: 13
```

Switching to WebSockets protocol
proxy.example.com -> Alice (TLS)

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: sip
```

SIP REGISTER request letting the server know of Alice's "location"
Alice -> proxy.example.com (TLS)

```
REGISTER sip:proxy.example.com SIP/2.0
Via: SIP/2.0/WSS df7jal23ls0d.invalid;branch=z9hG4bKasudf
From: sip:alice@example.com;tag=65bnmj.34asd
To: sip:alice@example.com
```

```
Call-ID: aiuy7k9njasd
CSeq: 1 REGISTER
Max-Forwards: 70
Supported: path, outbound, gruu
Contact: <sip:alice@df7jal23ls0d.invalid;transport=ws>
;reg-id=1
;+sip.instance="<urn:uuid:f81-7dec-14a06cf1>"
```

OK response
proxy.example.com -> Alice (TLS)

```
SIP/2.0 200 OK
Via: SIP/2.0/WSS df7jal23ls0d.invalid;branch=z9hG4bKasudf
From: sip:alice@example.com;tag=65bnmj.34asd
To: sip:alice@example.com;tag=12isjljn8
Call-ID: aiuy7k9njasd
CSeq: 1 REGISTER
Supported: outbound, gruu
Contact: <sip:alice@df7jal23ls0d.invalid;transport=ws>
;reg-id=1
;+sip.instance="<urn:uuid:f81-7dec-14a06cf1>"
;pub-gruu="sip:alice@example.com;gr=urn:uuid:f81-7dec-14a06cf1"
;temp-gruu="sip:87ash54=3dd.98a@example.com;gr"
;expires=3600
```

Once two peers have registered, one of them can send an invite to another through the server in the form of an INVITE request to form a signalling channel between them which WebRTC uses to establish a peer connection and data channel. To create this peer connection, it uses interactive connectivity establishment (ICE) to find a list of the possible IP's and ports of each peer which are then sent through the server. Once this is done, the data channel can be formed and packets can be sent peer-to-peer. These packets are managed and secured by Secure Real Time Transport Protocol (SCTP) for media and Stream Control Transmission Protocol(SCTP) for non-media and encrypted by Datagram Transport Layer Security (DTLS) [20].

Chapter 3

Specification

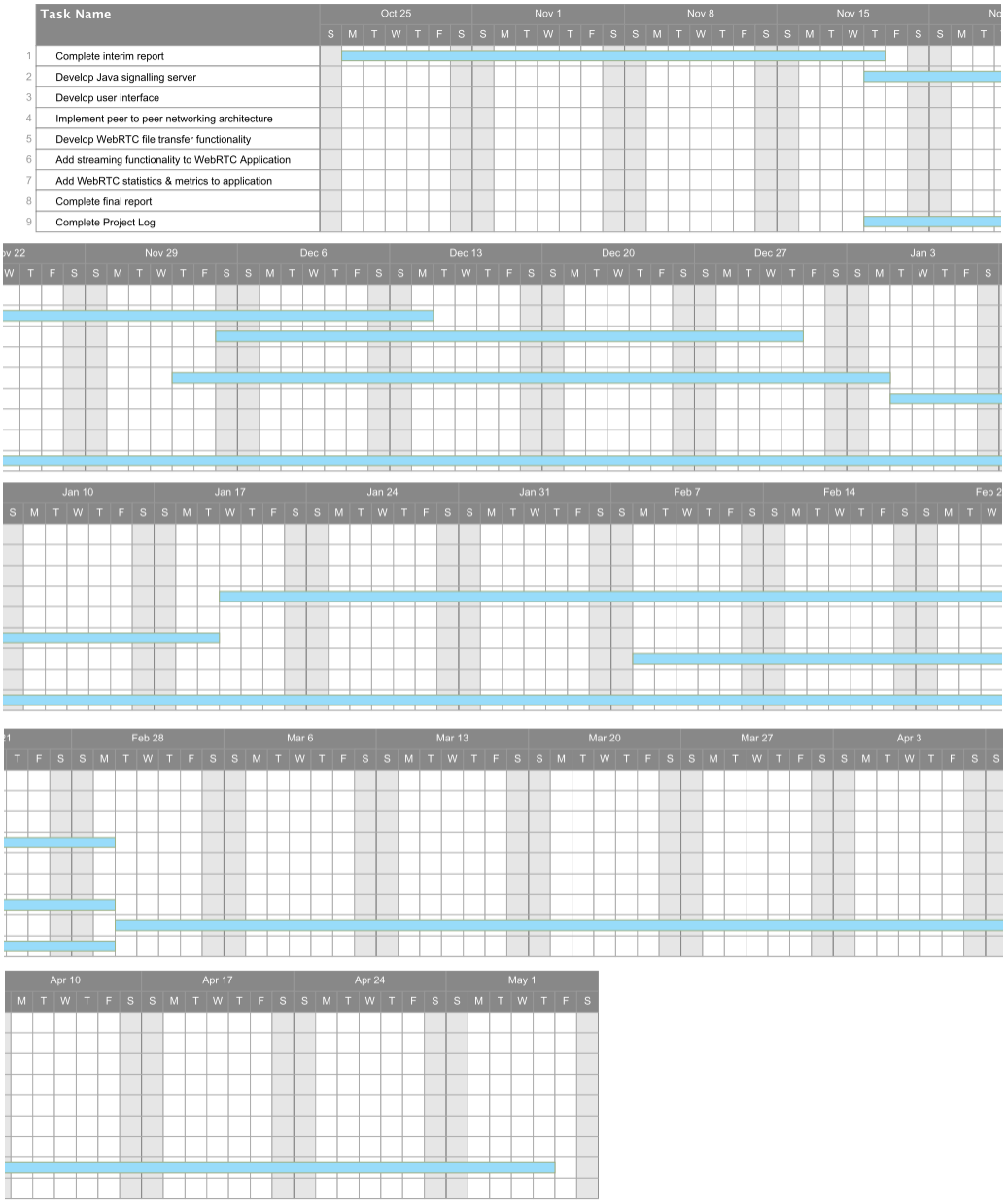
3.1 Deliverables

The first intermediate product will be the signalling server written in Java. This will handle the exchange of client meta data in order to establish the connection between two peers using the web application. I plan to overlap the development of this with the development of the data transfer functionality and user interface of my second deliverable, the web application as manually testing the signalling server will be a lot easier with a partially developed application to test it with.

The first end deliverable will be the client side web application the user interacts with in order to select a file as well as handling sending the meta data to the signalling server and managing the peer-to-peer data transfer and media streaming. This is broken down into several intermediate products, the data transfer functionality, the media streaming functionality, the peer-to-peer network and the user interface. I plan to produce the data transfer functionality first along side the user interface to allow for manual testing. After I have implemented data transfer, I will work on media streaming and forming the peer-to-peer network topology.

The second end deliverable will be the final report containing documentation and analysis using the metrics from my application, comparing how it and technologies behind it perform in comparison to others, focusing in particular on how peer-to-peer over the browser (WebRTC) compares to other methods of data transfer and media streaming.

Figure 3.1: Schedule of Activities



3.2 Risk Analysis

Risk	Probability (1-5)	Impact (1-5)	Mitigation	Contingency
Illness/Injury	4	3	Reserve time for illness Be hygienic Eat healthy Exercise	Allow time for recovery Take medicine to aid recovery
Inaccurate estimations	3	3	Be liberal with estimations Reserve time for deliverables behind schedule	Adjust scope of project
Data loss	1	5	Use a version control system Keep local backups	Recover data from Git
Uncommunicative stakeholder	1	3	Ensure regular meetings with stakeholder	
Stakeholder turnover	1	4	N/A (out of my control)	Get new stakeholders
Project scope too large	3	4	Research enough to be certain in project scope Be liberal with estimations	Adjust scope of project
Technologies too immature/insufficient for project	2	4	Research technologies beforehand	Find alternative technologies Adjust scope of project

3.3 Quality Analysis

The main measure of success will be the web application's performance in it's ability to transfer & stream data between two peers. To track this, I will implement metrics using the WebRTC "getStats" statistics API, which allows for monitoring of the bandwidth usage, packets lost and network delay within a data channel between two peers. I will then gather these statistics in two different cases, one where there is no formal network structure (to find a baseline) and then when the many-to-many network has been implemented to compare if the structured peer-to-peer network does improve performance of the application. This network will also have it's own monitoring to ensure that peers are being correctly organised. Further to this, I will also compare the speeds of file transfer with other services such as DropBox through their API, however this may prove unrealistic as these services will be running on more powerful servers. To certify that the user-experience is acceptable, the application will also be user-tested. The signalling server will be load tested in isolation to make certain it can handle multiple requests to ensure it does not bottleneck the application.

Chapter 4

Methodology

I chose to use an alternative methodology to the waterfall model because it lacks the ability to adapt to changes in a project deadline. Due to the way waterfall is structured into different phases that must be completed sequentially, often when changes such as new requirements occur, all these phases must be repeated in order to account for this. Iterative methodologies take an approach that can adapt to these changing requirements because they utilise short development cycles and focus on developing small modules of a product at one time, making it easier to revise a product if necessary. This is particular useful in my project as it is relatively experimental and the requirements of it may change regularly.

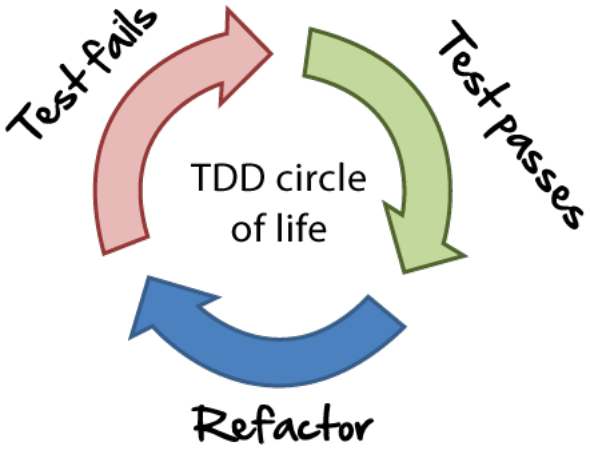
Thus, as a way of tracking the progression of my project, I plan to use an iterative and evolutionary methodology. This is used within software development as a way of incrementally developing applications through small cycles. Whilst it will be similar to scrum, it will not have the stricter framework surrounding it that requires a product owner and scrum master. This methodology will be relatively simple and based around a backlog of tasks from which a developer pulls from in a limited amount, normally 1 or 2 tasks at a time which will are then pushed through the development work flow.

In my case, the work flow will be relatively simple:

1. To Do
2. In Progress
3. Code Review
4. Manual Testing
5. Done

During the "In Progress" step of the work flow, I will use a test driven development (TDD) process in which tests are written first and then code is wrote to make the test work. However, I will be fairly lenient with this, only using this process on parts of the code that require stringent testing as writing unnecessary tests will take up development time.

Figure 4.1: Test-Driven Development (TDD) Cycle [21]



During the "Code Review" step, I plan to self-evaluate the task I have completed. On top of this, I will run static code analysis tools (such as FindBugs/PMD for Java) if the task is a coding task and use the code review section StackExchange to get second opinions. If it passes the code review step and it is possible to do so, I will black-box test the task from a user's perspective to see that it actually works as intended. Once every task forming a feature is completed, I will manually test the feature as a whole to see that each task has integrated together as planned.

To visualise this workflow, I will use the open source web application "Kanboard" which I have hosted on an AWS EC2 Instance. This will allow me to keep track of progress throughout the duration of the project. To manage the actual changes made to the project files, I have utilised the version control system Git with a private GitHub repository containing my project.

Chapter 5

Prototyping

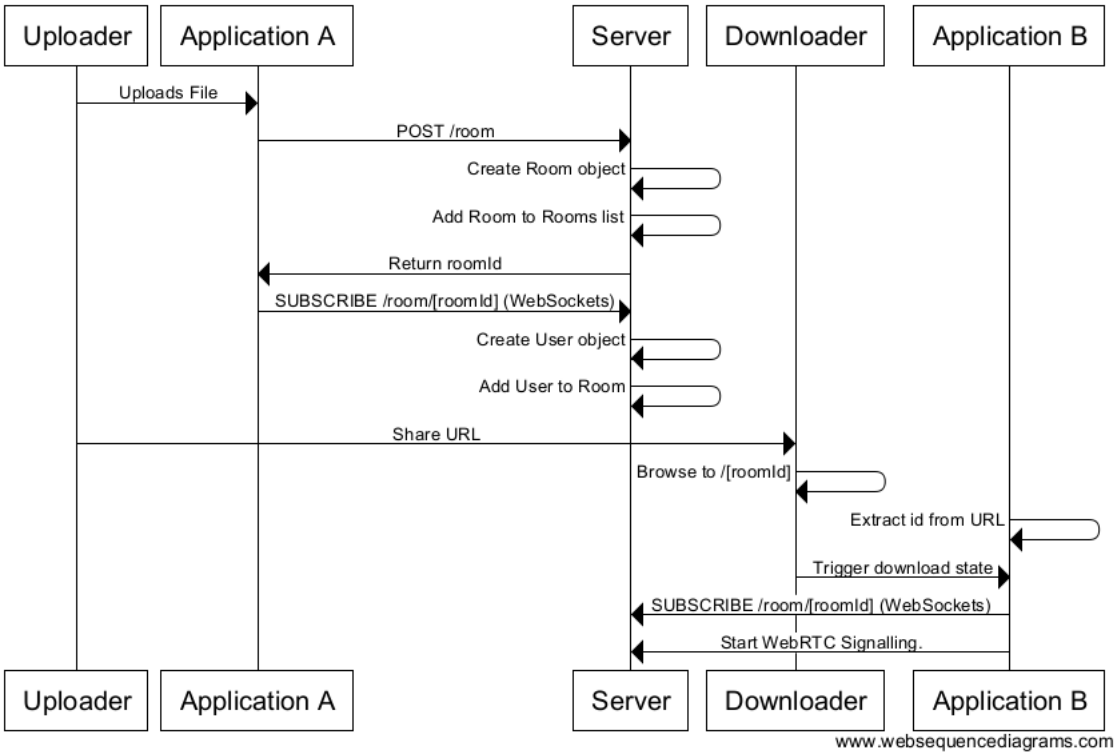
5.1 Signalling

5.1.1 1st Iteration - Spring Boot

The first prototype of the signalling server for Transfer4me was implemented using spring boot. This is a part of the Spring application framework emphasising convention-over-configuration, making it faster and easier to produce a working prototype. To support the WebSocket protocol, Spring uses the "STOMP" sub-protocol to model the message sent between server and client, STOMP is very similar to HTTP in how it models the message but is sent over TCP once the socket is established. Using the STOMP JavaScript client, an application can subscribe to several destinations and listen for events (by asynchronous callbacks) or publish events using a send method that takes a string identifying the event (which in this case is "signal") and another string representing the message. The architecture that was created on top of this protocols used rooms that were created when a user uploads a file designated via a URL which downloaders and streamers then subscribe to when they load the page.

To create the room that the users reside in, the client that uploads the file fires a request to the server using an endpoint independent of WebSockets. When the server receives this request, it generates a Room object and a random unique integer that serves as its identifier. This room is then added to a list of rooms and the Id is returned to the client in the response. The client then uses this id to subscribe to the room over WebSockets. When this connection is established, the server creates a user object and adds it to the correct room. This ID is then appended to the base URL of the page for the uploader to share. Once a downloader loads this url, they are redirected to a page representing the room and upon clicking either the download or stream button, the client will attempt to establish a WebSocket connection using the room id taken from the url.

Figure 5.1: Spring signalling establishment sequence



The main problem I found with the Spring implementation was in its limited functionality. It was extremely hard to implement sending messages to individual specific sockets which limited how multiple peer connections could communicate within the same room as they would all be receiving every signal being exchanged. This could be avoided by making each client application only use the messages that were destined for it by appending an identifier to each message but that would not stop the clients from receiving the messages, only how they used them. As a result, this is not a viable solution as it leaves the application open to man in the middle attacks as a user could easily modify the client-side application to log all of the messages that it received which is a major issue in this case where the the majority of the messages sent will be WebRTC metadata containing information such as your IP address.

5.1.2 2nd Iteration - Node JS

Socket.io

Dynamic Room Generation

Authentication

5.2 WebRTC

5.2.0.1 One-to-One peer connection

5.2.0.2 Using DataChannel to transfer files

5.2.0.3 Streaming media via peer connection

5.2.0.4 One-to-Many peer connections

5.2.0.5 Using GetStats library to measure performance

5.3 Interface

5.3.0.1 Experimenting with frameworks

5.4 Limitations & Improvements

5.4.0.1 Many-to-Many peer connections

5.4.0.2 Video Streaming

5.4.0.3 Authentication Encryption

5.4.0.4 Signalling server performance

5.5 Resources used

Chapter 6

Outcomes

6.1 Data

6.2 Discussion

6.2.1 Comparison to other services

Bibliography

[1] An Unstructured Peer-To-Peer Overlay Network. Available: http://www.hindawi.com/journals/jcnc/2012/485875/fig1/. Last Accessed 6 Nov. 2015.

[2] Eng Keong Lua Crowcroft, J. ; Pias, M. ; Sharma, R. ; Lim, S.. (Second Quarter 2005). A survey and comparison of peer-to-peer overlay network schemes. Communications Surveys & Tutorials, IEEE. 7 (2), 72 - 93.

[3] Yang, Beverly; Garcia-Molina, Hector (2001). Available: http://infolab.stanford.edu/ byang/pubs/hybridp2p_long.pdf. Very Large Data Bases. Last Accessed 06/11/2015.

[4] Schulzrinne, et al. (February 2010). Security in P2P Realtime Communications. Available: https://tools.ietf.org/html/rfc5765#section-7.2. Last accessed 05/11/2015.

[5] JOHAN GRONBERG, ERIC MEADOWS-JONSSON . (2014). Tree topology networks in WebRTC. Available: http://publications.lib.chalmers.se/records/fulltext/202811/202811.pdf. Last accessed 12/11/2015.

[6] Jan Sacha. (2009). Exploiting Heterogeneity in Peer-to-Peer Systems Using Gradient Topologies. Available: https://www.scss.tcd.ie/publications/tech-reports/reports.10/TCD-CS-2010-06.pdf. Last accessed 12/11/2015.

[7] Cloud Security Alliance. (2013). The Notorious Nine Cloud Computing Top Threats in 2013. Available: https://downloads.cloudsecurityalliance.org/initiatives/top_threats/The_N otorious_Nine_Cloud_Computing_Top_Threats_in_2013.pdf. Last accessed 09/11/2015.

[8] Greenwald, Glenn; MacAskill, Ewen (June 6, 2013). "NSA Taps in to Internet Giants' Systems to Mine User Data". The Guardian. Last accessed 05/11/2015.

[9] Andrew Griffin. (2015). Investigatory Powers Bill could allow Government to ban end-to-end encryption, technology powering iMessage and WhatsApp. Available: http://www.independent.co.uk/life-style/gadgets-and-tech/news/investigatory-powers-bill-could-allow-government-to-ban-end-to-end-encryption-technology-powering-a6725311.html. Last accessed 09/11/2015.

[10] WeTransfer. How long are my uploads available to download?. Available: https://wetransfer.zendesk.com/hc/en-us/articles/202603916-How-long-are-my-uploads-available-to-download-. Last accessed 05/11/2015.

[11] Amazon Web Services. WeTransfer Case Study. Available: https://aws.amazon.com/solutions/case-studies/wetransfer/. Last accessed 05/11/2015.

[12] N/A. A Study of WebRTC Security. Available: http://webrtc-security.github.io/#4.3. Last accessed 05/11/2015.

[13] Harald Alvestrand. (2011). Google release of WebRTC source code. Available: http://lists.w3.org/Archives/Public/public-webrtc/2011May/0022.html. Last accessed 29/10/2015.

[14] Adam Bergkvist, Daniel C. Burnett, Cullen Jennings, Anant Narayanan (until November 2012). (2011). WebRTC 1.0: Real-time Communication Between Browsers. Available: http://www.w3.org/TR/webrtc/. Last accessed 29/10/2015.

[15] Available: http://iswebrtcreadyyet.com/. Last accessed 30/10/2015.

[16] Mozilla Web API. Available: https://developer.mozilla.org/en-US/docs/Web/API. Last accessed 30/10/2015.

[17] Justin Uberti. (2011). Javascript Session Establishment Protocol (JSEP). Available: https://lists.w3.org/Archives/Public/public-webrtc/2012Jan/att-0002/JavascriptSessionEstablishmentProtocol.pdf. Last accessed 11/11/2015.

[18] Fette & Melnikov. (2011). The WebSocket Protocol. Available: https://tools.ietf.org/html/rfc6455. Last accessed 12/11/2015.

[19] Baz Castillo, et al. . (2014). The WebSocket Protocol as a Transport for the Session Initiation Protocol (SIP). Available: https://tools.ietf.org/html/rfc7118. Last accessed 12/11/2015.

[20] Jesup, et al. (2015). WebRTC Data Channel Establishment Protocol. Available: https://tools.ietf.org/html/draft-ietf-rtcweb-data-protocol-09. Last accessed 12/11/2015.

[21] Available: http://www.agilenutshell.com/assets/test-driven-development/tdd-circle-of-life.png. Last accessed 09/11/2015.