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

Dominic Rathbone

November 2, 2015

Contents

Todo list

Chapter 1

Interim planning & investigation report

1.1 Project Scope

Aims and Objectives

The ultimate objective of my project is to investigate the possibility of data transfer over web browsers (in particular audio and video streaming) without the need for a centralised client-server architecture, instead opting for a peer-to-peer architecture. In order to do so, I will need to research peer-to-peer networking as well technologies that will allow for it's development in the browser such as WebRTC. In order to test and demonstrate this objective, It will materialize in the form of a web application that people can use to both transfer and stream files (if in a suitable format).

Stakeholders

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

Methods of Communication

Stelios and I have set up a regular meeting once a week on Friday at 4pm to review progress and answer any questions. On top of this, we communicate regularly via email and I have set up a private git repository on GitHub to keep my project in which I will give Stelios access to.

Quality Analysis

Success will be measured by the performance of the web application's ability to transfer data. This will be achieved by application metrics recording how fast/reliably files are transferred under differing amounts of load. Load will be simulated using a load testing tool to mock client connections to the web application. I will implement metrics using the WebRTC statistics API, which allows for monitoring of peer connections.

1.2 Literature Review

Client-Server vs Peer-to-peer

Currently, The most common architecture for file storage/transfer and video streaming services is the client-server model. This is an network architecture in which the machines doing the data processing (servers) are separated from the machines running the application (client). 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, if these servers becomes congested as load increases or in extreme cases, if the servers go down, the user would not be able to use the application as intended. Using a decentralized peer-to-peer networking architecture avoids both these issues as processing is distributed between the peers making up the network, resulting in it being cheaper to maintain and scale the service as load increases whilst being inherently robust.

in my application

Security is a concern with peer-to-peer networking as there is no gatekeeper stopping clients from connecting to your machine whereas with client-server architecture, the server can demand authentication and authorization from the client. Security issues in peer-to-peer could allow for unwanted access to your private information. Whilst it would be hard to mitigate risks like this from my position, I think allowing users of the application to enable authentication (via password or keys) on the page where the connection lies would give them a level of access control that could avoid unwanted access.

Peer-to-peer networking topology

The entire idea behind my application is based on this peer-to-peer connection between two browsers. Whilst this one-to-one (unicast) connection is formed and maintained by WebRTC, it does no more than this. If I want to form an actual many-to-many (multicast) network of peers then I will have to produce code that will discover and co-ordinate these unicast connnections.

A network can be structured in different ways, this structure is referred to as it's topology. There are several different network topologies but I plan to focus on mesh networking as I feel it would best represent the structure that the network of peers using my web application would form. A mesh network is:

WebRTC

WebRTC 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 them. This was first released by Google as an open source project to implement real-time communication in the browser in May 2011 [1]. It was later drafted as an API definition by W3C which is still a work in progress. [2]. WebRTC has yet to be fully implemented in every web browser but Chrome, Firefox and a WebRTC specific browser called Bowser are leading the way in implementing the API definition. Firefox (Nightly) seems to be leading the way in this so I will focus specifically in developing towards this browser[3]. 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." [4]
- 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." [4]
- 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." [4]

My Web 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.

Although WebRTC is used to transfer data, it purposely doesn't handle signalling. Signalling is a concept that came from telecommunications and VoIP. It is the process of organising the communication between two clients and handles the exchange of metadata that creates and manages a session between two clients. With WebRTC, the most prevalent signalling protocol is the Session Initiation Protocol (SIP). SIP

This data is sent over a protocol called Secure Real Time Transport Protocol (SRTP). This protocol was originally developed for secure VoIP but has been adopted by WebRTC.

Media streaming

BackBoneJS framework

1.3 Specification

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 many-to-many networking functionality 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.

In terms of risk, I think this will be the largest in my project as it is the only tangible end product and it relies on WebRTC which is a relatively immature technology. As it is new (released in 2011) and it is the only technology at the moment enabling browsers to communicate peer-to-peer, it has not truly been tried and tested. This means there is a higher potential for problems such as security flaws to exist.

1.4 Methodology

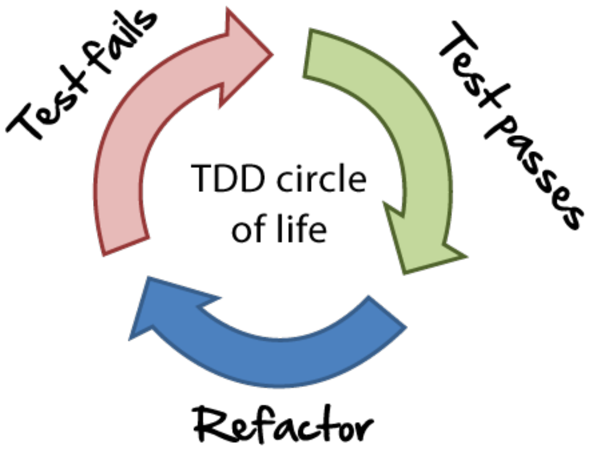
As a way of tracking the progression of my project, I plan to use Kanban methodology. This is used within software development as a way of incrementally developing applications through iterative cycles. Whilst it is similar to scrum, it does not have the stricter framework surrounding it that requires a product owner and scrum master. It also avoids overloading developers with restrictive time-boxed sprints. This methodology is relatively simple and is 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. Development iteration per task

- Coding
- Code Review: Code will be reviewed by self-evaluation, static code analysis and reviews from other peers and StackExchange.
- Manual Testing: If applicable, black box testing will be used to evaluate the code from a user's perspective.

2. Stakeholder Approval/User Acceptance Testing: Once all the tasks producing a working feature have been completed, it will be tested.

During the coding period 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.



Tools and Environment

Git IDE Unit testing framework build tools

Bibliography

[1] 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.

[2] 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.

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

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

[5]