**Dependencies**

* Nodejs
  + Dgram module (udp sockets)
  + Node-speaker (for audio out)
  + Stream

**Notes**

* Going to implement asynchronous functions to allow for the requesting, processing, and receiving of lost and received packets from the server.
* Or I could use the library “async” that allows me to use multiple processes to request for missing chunks or just handle the main queries
* Need to find a way to implement concurrency with async
* There is no multithreading in JS nor is there proper way to create multiple processes.
* ~~Need to implement the sending of the packets inside the 2d filebuffer array in the server.~~
* Also need to implement the protocol that will handle all of this.
* I need to implement the client’s side who will receive the file packets in partitions.
* A readstream is used to buffer the data and will pipe to the speaker and the output file. If the data was written directly to the file, it would not be possible to write data to the speaker, or it would require the use of another read stream that would read the file that is being written to. This implementation would cause a number of problems as downloading and writing to the file is slower than using a readstream to read the data first. Implementation a would cause the readstream to read the outputted data faster than it can be downloaded which causes the audio to cut out faster than it should.

**Protocol**

* Start transfer
* Initiate File transfer
* Partition Packet
* Initiate Transfer
* Partition Finished
* File Transferred

**Introduction**

The aim of this practical is to create a client/server application that allows for the streaming and transfer of an audio file over UDP as the transport-layer protocol, while at the same time testing and documenting how the application deals with the transfer of data over a lossy and unreliable channel. To satisfy these requirements, JavaScript (and a few supporting libraries) was used to create and implement an application layer that supports reliable streaming and transfer of data over different kinds of networks.

**Design**

1. *Protocol*

*Protocol Messages*

The protocol used by both the server and the client is designed in such a way that certain actions will only be performed depending on the type of protocol message that is found in the header of each packet. These messages allow the client and server to decide what the data in the packet will be used for and how the data is structured. Following this format, the partition-based method of transferring packets was successfully achieved. This method allows the file to be divided into percentages known as partitions which each contain a uniform number of packets. Once the packets are divided into percentages, the file will be transmitted a percent at a time, allowing each percent to be received and analyzed for missing packets. Only after the partition has been cleared of any errors will the next partition be requested from the server. This process is then repeated until the entire file has been transferred.

* 1. *Packet Structure*

1. *Buffering*

**Implementation**

1. *Dependencies*
2. *Protocol*
3. *Buffering*

**Testing**

**Conclusion**