Mustache Messaging App

Sending messages reliably using UDP

Gribble, Hudson  
Computer Science  
 University of Memphis  
 Memphis, TN  
 [hgrbble1@memphis.edu](mailto:hgrbble1@memphis.edu)

Ault, Kyle  
 Computer Science  
 University of Memphis  
 Memphis, TN  
 [kault@memphis.edu](mailto:kault@memphis.edu)

ABSTRACT

Our project is a messaging app that offers reliable data transfer using UDP protocol. It uses peer-to-peer architecture between multiple clients. Reliable data transfer is achieved by initiating a handshake and then beginning Go-Back-N protocol, sending the same handshake message and data packets to the full list of desired recipients. In order to reassemble the file, message, or image sent, the receiver uses headers that are wrapped with the data to be sent by the sender.

Electron, which uses Node.js, is the framework we used for our program. Using Javascript and HTML makes for an easy and polished user interface experience.

1 The Mustache Messaging Application

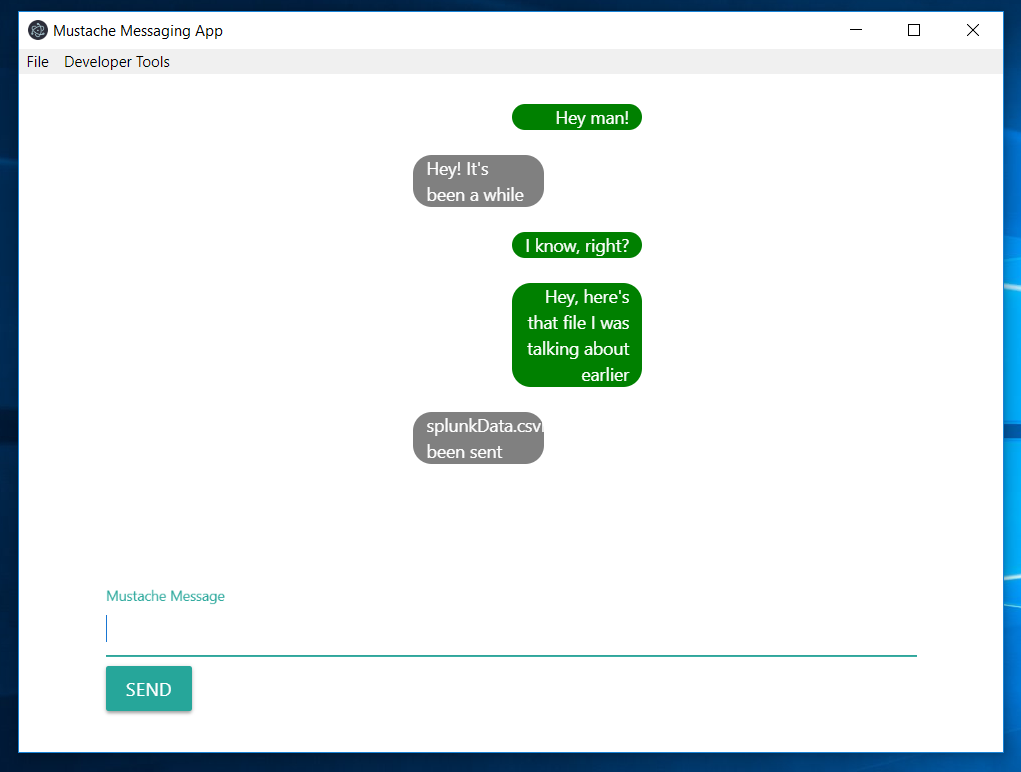


Figure 1: Our UI, with an example conversation

The Mustache Messaging Application’s name is rather arbitrary – we chose it to add a little bit of fun and flavor to an app that has been made in many different ways.

If we had more time, this unusual name was to become a theme of the entire application – even with little pictures of mustaches inside the text bubbles. We felt that a hook like this to distinguish our app was important, because there are so many messaging apps out there. As it is, our UI is relatively simple.

1.1 Problem Motivation and Design Goals

Our main goals for this project were fourfold:

1. Support file transfer up to at least 1 GB at a reasonable speed
2. Have a UI that is intuitive and non-technical
3. Implement Go-Back-N protocol to ensure reliable data transfer
4. Allow multiple clients to chat at the same time

We hope to gain a better understanding of Go-Back-N protocol, as well as UDP and file transfer in general as we finish this project.

1.2 Approaches and Techniques

Our entire application is driven by the events of receiving messages on the UDP sockets that each client is running. Aside from starting the process with a method call initiated by the UI, each file transfer runs by responding to one of 4 different packet types we created: handshakeInit, handshakeAck, data, and dataAck. The sender deals only with sending handshakeInit and data, while the receiver sends a handshakeAck in response to a handshakeInit packet and a dataAck in response to a data packet.

When the user initiates the sending of a file, the file is sliced into segments and stored into an array of packets. Each data segment gets some headers added before it is wrapped into a packet: fileName, segmentNumber, numSegments (total number of segments in this file), and filetype (‘image’, ‘file’, or ‘message’ – all three fileTypes are sent the same way, it is only the UI that differentiates between them.

Whenever a client receives a data packet, it appends that segment to the file that it has been writing and then discards the data from the packet. This reduces the data needed in RAM to receive a file, meaning a client can receive massive files.

To send to multiple clients, the user must change the TARGETS variable in the code to be a list of addresses and ports to send messages to. Then whenever a user sends a message, it will be sent to every listed address and port. To keep track of files being sent and received from different clients, the program uses a dictionary that looks up the file name, address, and port attached to the packet it received, and continues from the same spot in the file sending process.

In order to send a message, the program saves whatever is typed as a message to a text file, and then segments it and sends the text file just as it would send any other file. This means that there is technically no limit to the length of message that can be sent.

The Go-Back-N protocol worked by having the sender send a window of size N packets (set by a variable), and then set a timer for the timeout. If no new acks were received by the timeout, then the whole window would send again. If a new ack was received, the sender would increase the window starting point and send the next packet, and then reset the timer. Whenever the receiver receives a data packet, it will send an ack for the highest segment number it has received so far.

On close, every text in the window is saved to a file called history.json, so that the message history can be pulled up the next time the user logs in.

Because our interface supports UTF-8 encoding, emojis can also be sent from client to client, as long as a user copies and pastes them in from the web.

1.3 Evaluation Plan

In order to evaluate our project, we will focus mostly on three areas that we think are important:

1. Reliability – the program should be able to send an intact file even when the network is dropping packets. To test this, we will send files to multiple clients, and set the receivers to drop 1 out of every 100 packets, to see if the file still gets there intact. 1 out of 100 sounds small, but for a file with 20,000 segments, the drops add up.

2. Speed – The program should add as little delay as possible, because there will already be significant delay due to sending data over the connection between the clients. We mostly want to measure the speed of our program without adding in network delays, so we will send files between processes running on the same computer. We will also test the speed between machines.

3. Max file size – We think the file size will be limited by the RAM of the sender. Because the receiver writes to file and then discards the packet upon receipt, it may not have a limit. The sender, however, reads the whole file into RAM and then sends it.

We will also test the chat between multiple clients, to make sure that sending multiple files doesn’t cause errors.

1.4 Evaluation Results

Even when our receiving clients were failing to receive packets occasionally, the Go-Back-N protocol ensured that the dropped packets were repeated. To repeat our tests, simply change the dropPackets variable in our code to true, and try to send a file.

Multiple clients were able to chat with each other, and each client received all messages sent from the other clients. We tested this be sending messages between three clients.

As for transferring files, we tested the speeds and the max file sizes together. The largest file we sent was 1GB, but with more RAM, bigger files could have been sent. Our results are summarized in this chart.

|  |  |  |
| --- | --- | --- |
| File Size | Time to transfer - one machine (no network delay) | Time to transfer - between machines |
| 1 KB | 21.9 ms | 83 ms |
| 2KB | 23.3 ms | 93.61 ms |
| 4KB | 22.9 ms | 115 ms |
| 64KB | 55.9 ms | 211.626 |
| 1 MB | .425 s | 1.2s |
| 10 MB | 3.137 s | 11.0 s |
| 100 MB | 27.851 s | 116.7 s |
| 500 MB | 281 s | -- |
| 1 GB | 658 s | -- |

(ms = milliseconds, s=seconds)

1.5 Conclusions and Future Work

Electron turned out to be a great framework for an app like this. It has a good UDP socket library, and the HTML-Javascript combination meant that the UI was easy to make.

The Mustache Messaging App could still use more polishing. There are a few errors with file naming that we didn’t quite fix, as well as the occasional message repeated in the chat. Many little features to enhance the user’s experience could be added also, such as usernames that link to specific IP addresses and ports, and quick add buttons for emojis.

If we were to rewrite the sending methods to send data segments as they are read from the file, that file size limit would go away entirely, and we would be limited only by our patience in the size of files we wanted to send.

Overall it is an app that does what it was intended to do. it offers reliable, quick data transfer for files up to a few gigabytes between multiple clients. We are happy to report that the Mustache Messaging is what it claims to be – a real messaging app.

ACKNOWLEDGMENTS

We studied Go-Back-N protocol largely using the interactive tool made by Pearson Education, which can be found at the URL below.

https://media.pearsoncmg.com/aw/ecs\_kurose\_compnetwork\_7/cw/content/interactiveanimations//go-back-n-protocol/index.html

Conference Name:ACM Woodstock conference

REFERENCES

None cited

Conference Short Name:WOODSTOCK’18

Conference Location:El Paso, Texas USA

ISBN:978-1-4503-0000-0/18/06

Year:2018

Date:June

Copyright Year:2018

Copyright Statement:rightsretained

DOI:10.1145/1234567890

RRH: F. Surname et al.

Price:$15.00