Janus: A Decentralized Shipping Network

Of the people, by the people, for the people

Sachal Malick

CSE 426: Blockchain Applications

Fall 2020

Professor: Dr. Bina Ramamurthy

Teaching Assistant: Arnab Das

Table of Contents

Introduction:

In recent months the United States Postal Service has faced serious financial hardship resulting in heated political battles over the responsibility of the Congress to continue and expand its funding. A public shipping network is one of the most essential services a government should provide for its citizens, so citizens can have a reliable, affordable, and secure means of communication and transportation. With the uncertain future of the USPS in mind, we sought out a way to harness the power of decentralized technology to create a shipping network that is by the people and for the people.

Our solution is Janus, a decentralized shipping service that allows for anyone to ship a package to any location through a network of disintermediated drivers that anyone can join or leave as they please.

Specifications

Blockchain: Ethereum
Network: Ropsten
Language: Python
Web Framework: Flask
Address format: geographic coordinates: [longitude, latitude]
Max shipments per customer: 1
Max cargo for drivers: 1 package per trip

Features:

Customers:

-Create a shipment for pickup at any origin location for delivery to any destination
-View cost of shipment (not accounting for gas fees) which varies according to distance from origin to destination.
-View status of current shipment
-View identity of driver carrying shipment when package is picked up

Drivers:

-Toggle your status between “Available for driving” and “Unavailable”
-Claim a package for transport when you would like to work
-Register a package as delivered when you drop it off at the destination address.
-Receive payment when the package is delivered.

User instructions:

For Drivers:

1. Follow the instructions in README.md to install the required dependencies and run the dapp.
2. Open a browser window and navigate to localhost:8080
3. Enter a decentralized identity in the field that says “enter your address”, then click Ship a package.
4. Enter your private key in the first field:
    a. To find your private key in meta mask
    b. Click on the metamask icon in your browser
    c. Click on the account you would like to use
    d. Click on the three vertical dots
    e. Click “Account Details”
    f. Click “Export Private Key”
    g. Enter your account passcode
5. For the home address: enter any latitude longitude with values greater than 0. 3,3 or 8,4 is a good example. They should be separated by a single comma.
6. For the remaining values enter an integer greater than 0 and less than 500.
7. Click the Sign Up Button
8. When you are successfully registered you will be supplied with a link, click on it
9. In the text field, enter your private key then click “Become Available”
10. Once you are available, enter your private key and click “Claim Package”
11. Once you have picked up the package and delivered it to the specified address, click Drop Off Package.
12. The funds for the drive will be delivered to your account.

For Customers:

1. Follow the instructions in README.md to install the required dependencies and run the dapp
2. Open a browser window and navigate to localhost:8080
3. Enter a decentralized identity in the field that says “enter your address”, then click Ship a package.

13. Enter your private key in the first field:
    a. To find your private key in meta mask
    b. Click on the metamask icon in your browser
    c. Click on the account you would like to use
    d. Click on the three vertical dots

    e. Click “Account Details”
    f. Click “Export Private Key”
    g. Enter your account passcode

14. For the home address: enter any latitude longitude with values greater than 0. 3,3 or 8,4 is a good example. They should be separated by a single comma.
15. For the remaining values enter an integer greater than 0 and less than 500.
16. Click the Ship button
17. Once your order is processed you will see a link that you should follow to view the status of the shipment.

Test Plan:

Test case 1: valid run

1. Create shipment with address 1, ensure address 1 has balance of at least .01 ethers + the estimated cost of shipping
2. Status should say “awaiting pickup”
3. Sign up driver with address 2
4. Set driver as available
5. Claim package
6. Driver should receive package with address 1
7. Refresh shipper home page: should see address 2 and status that says “in transit”
8. Driver should drop off package
9. Refresh shipper home page: should see status that says “delivered”

Test case 2: insufficient funds

1. Attempt creating shipment with address that has a balance of .01 wei
2. Create a shipment from 1,1 to 200,200
3. An insufficient funds notification should be visible.

Challenges:

I was initially planning on developing the Dapp with Node.js, but after discovering that there was a well documented web3 library for python, I decided to try giving it a shot using Flask as my web framework. While I learned a lot from struggling through a library that's actually not frequently maintained and has some significant gaps, it did slow down the work flow significantly. I frequently found myself in a position where I had to edit core library code just to get more descriptive error messages.

Working locally with ganache in many ways mirrored the dapp's we worked with in lectures, however interacting with infura was a whole different paradigm. I was only able to work with raw transactions which means I had to rewrite a lot of the code that was using library functions that were just delegating to RPC methods.

I was still in a traditional programming mindset when I began the design process so a lot of my early ideas for the contract had to be splintered. I ultimately ended up using a simple priority queue over a linked-list esque structure, but rather than having each link correspond to another object, it corresponds to the key that maps to the object.

Future Developments

I'm really disappointed that I didn't have time to expand on my user interface. Ultimately I spent about a total of 2.5 hours on it compared to the contract and the web3 connection point. There's a lot of features that could heavily improve the interface right now that I could implement with another hour of development time, such as displaying more information relating to the package, and using events to trigger refreshes.

I definitely want to use IPFS and setup an API on a cloud server for computing between links so I can optimize the package matching process. That way I could allow for the driver to control how far they want to travel, and have another driver pick it up from their dropoff location. I learned a lot about the differences between decentralized and centralized programs through trying to tackle this problem. In a traditional paradigm you could easily calculate the entire route before travelling. In this paradigm I would use the same heuristics and path finding algorithms, but I would be making each decision as the package travels from driver to driver.

I also wanted to incorporate the capacity of the driver's vehicle and the size of the packages so I could allow for drivers to carry multiple packages at the same time. Again, this would've required heavier computations that I would have had to delegate to a cloud server.

Acknowledgements

I would like to begin by thanking Dr. Ramamurthy, who was a tremendous guide into the world of blockchain application development. Our future is decentralized, and thanks to this course I've been able to begin my exploration of this territory while it's still unsaturated and full of opportunities to have a meaningful impact.

Additionally I would like to thank the entire teaching staff, with special gratitude to the teaching assistant who I chose as my mentor for this term, Arnab Das.