Building a Decentralised Taxi App

Scott Street
159027651

April 2018

Abstract

Decentralised apps are pretty cool, but they don't quite work the same as what we're used to.

Contents

1 Introduction

Taxicoin is an attempt at designing and building a user-facing application for hailing taxis, where the entire system is fully decentralised, with no central authority in control. The motivation behind this is to combat some of the issues found with existing similar traditional applications, such as *Uber* and *Lyft*.

These companies saw a way to improve the taxi industry, and by improving the user experience and ease of ordering a taxi, attracted many users. However, as was the case with existing taxi companies, they still take a significant cut of fares. Coupled with the fact that they attempt to keep fares lower for passengers, the drivers are left with very little earnings.

2 An Explanation of Ethereum

Ethereum is a platform consisting of three components: Swarm, a "distributed storage platform and content distribution service" [1]; Whisper, a peer-to-peer communication protocol [2]; and the Ethereum Virtual Machine (EVM) used for running smart contracts [3]. The latter is often referred to alone simply as "Ethereum", however all three should be considered part of the same platform, each one complimenting the others. The aim of this section is to explain how these three solutions are used together to develop fully decentralised applications, or dApps.

2.1 Blockchains

The EVM component of Ethereum is built on top of a "blockchain", a term coined by the anonymous creator of Bitcoin [4], which is the original, and most widely known application of such technology. At its core, a blockchain consists of transactions, grouped together into "blocks", with each group also referencing the previous one, thus forming a "chain". A blockchain can be thought of simply as a form of database, keeping a state. A transaction represents a state transition, but must be verified before being deemed to be valid and placed in a block.

The blockchain itself is distributed across all nodes in the network (except in cases where a node chooses to reference another's copy), meaning that, unless explicitly obfuscated by the user, all transaction data on the network is open. This allows the auditing of transactions by any node on the network, and eliminates the need to trust a single entity to provide accurate data - this is the concept of trustlessness.

Transactions on the Bitcoin blockchain are, for the most part, simply that - transactions. They represent a transfer of funds from one "address" to another. They can additionally contain an amount of data, representing anything from a simple message, to a method call in cases where the receiving address is a "smart contract" (or simply "contract").

In Bitcoin, contracts are a special type of address which causes nodes on the network to execute some predefined code when a transaction is sent to it. Contracts are deployed by a standard (human controlled) address, but once deployed act as independent entities. Unless their code contains functionality to do so, the deployer has no control over the contract.

However, contract execution on the Bitcoin network is not Turing complete, due to the halting problem - the inability to determine whether a section of code will complete execution without looping infinitely. If contract execution was Turing complete in the existing Bitcoin network, a

malicious actor would be able to perform a denial of service attack against the network by deploying and calling contact code containing an infinite loop.

This is where the EVM differs, with the addition of "gas". This introduces a fee per instruction to be executed (paid in Ethereum's native cryptocurrency, Ether). The sender of a transaction sets the maximum amount of gas they are willing to spend for a transaction to complete, and a contract call will continue executing until either the execution completes, or the maximum amount of gas is consumed. This safely allows the use of loops within contracts, as it becomes very expensive to perform an infinite-loop attack.

The result is that the EVM is Turing complete, and thus in theory any arbitrary program can be implemented in a contract, opening the door to a wide variety of applications.

2.2 dApps

While smart contracts are well suited to taking inputs, making state changes, and producing outputs, that is all they do. It is possible to interact with them via a command line interface, through an Ethereum node, however this is obviously far from the desired experience for end users. To address this problem, several attempts at providing a user interface layer for the Ethereum network have been introduced. The most widely adopted, and officially endorsed solution, is Web3 - a browser API which allows interacting with all parts of the Ethereum platform from Javascript embedded on a web page.

This leads to the approach that many Ethereum dApp developers take: considering their application as a traditional "Single Page App", where instead of calling HTTP API endpoints, they are now interacting with a smart contract through the use of Web3. Smart contracts effectively take the place of a "backend" web server, leading to many benefits over traditional web apps, such as availability, security and integrity.

Coupling this with Whisper allows for peer-to-peer communication between instances of a dApp, which in most cases translates to between different users. For example, two parties negotiating the price of an item to be purchased - they do not necessarily want their negotiation to be public (or rather, it does not provide any value for it to be), therefore they can come to an agreement "off-chain" before publishing (sending) a transaction of the final agreed price. Whisper is also beneficial for situations where the sender of a message wishes to remain anonymous. When publishing a transaction on the blockchain, the sender is published along with it, whereas in Whisper, unless signed, it is improbable to determine the sender of a message [cite].

Additionally, the static HTML, Javascript and any additional components of a dApp can be hosted from Swarm (or a similar platform such as IPFS [cite]). When a file, or set of files, is published to Swarm, a hash is computed, and the file is split into pieces called "shards". The shards are then distributed across nodes in the network, with the intention that if one node becomes unavailable, the shards of the file should still be accessible. When a user wishes to retrieve a file at a later date, they can provide the previously computed hash to a Swarm node, which will request shards of the file from its connected peers.

In this manner, it is possible with the Ethereum platform to develop fully decentralised applications where the user interface is written as a

web page and is served from Swarm, thus eliminating the requirement for a traditional web server. An application's "backend" logic is contained within a smart contract, removing the need for a backend web server such as PHP. And finally, instances of the application may communicate between each other through the use of Whisper, removing the need for a solution such as WebSockets, where a central signalling server is required.

2.3 Smart Contract System Design

As smart contract execution is only ever triggered as a result of a transaction, applications must be designed around deliberate actions. For example, where in a traditional system, a method may be set to execute at a particular date and time, in a smart contract this is not possible. Instead such a method may only have a check for if the allowed time of execution has passed, and must be manually triggered by a transaction.

As the reasons for some of these differences are unlikely to be clear to users, it is important to consider how to communicate them.

Additionally, as there is an attached "gas" fee for publishing transactions and calling contract methods, it is in the interests of the users for the contract developer to make contracts as efficient as possible, and to make a minimal number of contract calls in a dApp. One way of doing this is to avoid on-chain interaction wherever possible, through the use of peer-to-peer protocols, predominantly Whisper. In extreme cases, complex routines within contracts can be written in the underlying EVM byte-code for improved efficiency.

2.4 Benefits of Decentralisation

3 Identification of Requirements

The first step towards developing Taxicoin was to identify the requirements for the resulting system. These were split into two categories: minimum viable and additional requirements.

3.1 Minimum Viable Requirements

These requirements are those which must be included in order for the system to correctly function.

Drivers must be required to pay a deposit in order to advertise to act as a reasonable barrier to entry. Without this in place, the network is easily open to spam and scammers posing as drivers. The deposit acts as an incentive to behave well.

Riders must advertise jobs to drivers on an individual basis in order to protect the privacy of the rider. As this is likely to contain individually identifying information, such as location, if this were published it could be used to track an individual.

The fare must be determined by quotes from driver to remove the need for a centralised fare decision. This is due to the fact that the fare depends on many factors which cannot be automatically determined in a decentralised and reliable manner, such as distance and demand. The

alternative would be fixed fares, but this is highly undesirable as short trips would be overpriced, and long trips underpriced.

Riders must pay fares to a contract in advance as a security measure, due to the fact that there is no other way to guarantee riders will pay after the fact. Without this, it is likely that a subset of riders would not pay for journeys.

Riders must provide an additional deposit before starting a journey to act as an incentive to successfully and formally complete a journey in the system. Without this, riders may have paid fare and have no regard for consequences of bad acts. Additionally, they may not carry on to rate the driver, an integral part of the smooth running of the system.

Riders and drivers must both rate the other on completion of a journey to affect the reputation of the other party. This is likely to be implemented such that a user is unable to interact with the rest of the system until they have formally completed their previous journey. Without this requirement, there is no way to determine the trustworthiness of another individual on the network, which is key to preventing bad behaviour.

When a journey is completed, deposits should be returned to the respective parties, and the fare paid to the driver this ensures that riders and drivers both have a stake in formally completing a journey. If they do not, their deposits are not returned, and neither is the driver paid. Without this deposit system, there is no guarantee that either party will rate the other - potential hit-and-run scenarios could occur where a rider uses the system only once and does not care to formally complete a journey and rate their driver as it provides no benefit to them. With the deposits however, they are likely to complete the process, at stake of losing their funds.

3.2 Additional Requirements

These are requirements deemed as "nice to have" features, without which the system will continue to function, but the addition of which would improve the system in some way.

Prospective drivers and riders should be able to informally communicate before forming a contract to allow any additional requirements on either part be known. For example if a rider is wishing to take a large, bulky item on the journey with them, they may communicate this in advance. If it transpires that the driver's car is small, the journey can be cancelled (or not formally begun), and another driver arranged, before the original driver has taken the effort of travelling to pick up the rider.

Dispute resolution should be built into the system for situations where driver and rider are unable to successfully complete a journey. This would work in a similar way to negotiating a price. In a worst case scenario, the driver wants payment in full, but the rider wants to pay nothing. In this case, the two negotiate until an agreement is reached. If they do not reach such an agreement, it reflects poorly on both parties, as the fact

that they have an unresolved dispute is public. The system is able to function without this, but bad disputes are likely to go unresolved which is dissatisfactory.

4 Protocol Specification

The protocol portion of Taxicoin is designed to be open. As such, anybody should be able to implement it in their own software. The following section of this document should be sufficient to do so.

4.1 Methods

Each of these methods is intended to be part of a smart contract. When one is called, it will modify the state of the contract, and/or return a value.

The specified arguments are to be supplied when calling that function of the contract. The *payable* keyword indicates that a method accepts a transaction with a currency value attached.

4.1.1 Driver Advertise

Arguments latitude; longitude; Whisper identity

Payable driver deposit value, defined in contract

The advertise method takes a location and deposit (value as defined by the contract settings) from a driver and the contract publishes the location of the driver.

If a deposit has not already been provided, and is not sent with the advertisement, an error is thrown. If deposit is sent, but has already been provided, the excess is returned and the method returns successfully.

4.1.2 Driver Advert Revoke

Arguments none

Payable no

If an active advertisement exists, its state is set to invalid, indicating that riders should not consider this driver. Deposits are not returned as a result of this action.

4.1.3 Rider Create Journey

Arguments driver address, fare

Payable fare plus rider deposit value, defined in contract

Accepts a quoted fare for a journey as a rider and forms contract between driver and rider, taking full fare plus deposit from rider.

The contract is not complete until the driver formally accepts the job by calling the accept journey method. Before this happens, the journey may be cancelled with no adverse effect for the rider, with the fare and deposit being returned in full.

This is intended to be called after an off-chain negotiation, with job proposal and quote messages.

4.1.4 Driver Accept Journey

Arguments rider address

Payable no

Formally accepts a job, committing both the rider and driver to its completion (or otherwise amicable resolution) at the stake of the fare and deposits.

If the rider, as identified by the given address, has not initiated a journey contract, then the method will return an error.

4.1.5 Complete Journey

Arguments none

Payable no

Marks the current journey as completed, as either the rider or driver. The journey will not be fully complete until both parties have called this method.

Once the journey is complete, the fare is paid from the contract to the driver, and deposits are returned to both parties.

4.1.6 Cancel Journey

Arguments none

Payable no

Proposes the cancellation of a journey, or in the case that the other party has already proposed a cancellation, accepts the proposal.

The fare is returned from the contract to the rider, along with the rider's deposit. The driver's deposit is not returned.

4.2 Messages

Driver and rider user clients should be listening for the following messages, where applicable. These messages are communicated via the Whisper protocol.

Message topics are always a length of 4 bytes (4 ASCII characters), therefore any topics listed here of a length less than 4 byes are right-padded with spaces.

4.2.1 Job Proposal

Topic job

This message is sent by a rider to a prospective driver, indicating that they wish to make the described journey. It is intended to be sent to advertised drivers matching a specified criteria, e.g. within a certain distance, with at least a certain reputation. However the sending of these messages is not intended to be carried out manually by the user – rather there is an automated process which fetches the list of active drivers and determines which to propose to.

The payload consists of an ASCII string of a stringified JSON object containing `pickup` and `dropoff` locations, as well as the network (Ethereum) address of the rider.

```
{
  "pickup": {
    "lat": String,
    "lon": String
  },
  "dropoff": {
    "lat": String,
    "lon": String
  },
  "address": String
}
```

Should a driver be interested in a proposal, they respond with a quote message.

4.2.2 Driver Quote

Topic quot

This message is sent by a driver as a response to a job proposal. It contains the network address of the driver, as well as the fare for which the driver is willing to take on the job. At this point, the quote is not binding.

```
{
  "address": String,
  "fare": Integer
}
```

If the rider chooses to accept the quote, they next call the create journey method.

4.3 Process Flow Diagram

5 Implementation

My implementation is in two parts: that of the protocol described above, and additionally an example *Web3* client.

6 Testing and Validation

6.1 Unit Tests

6.2 Static Analysis

References

[1] Viktor Trón. *What is swarm?* 2015. URL: https://github.com/ethersphere/swarm/blob/ac8b54726551d7a590f85d6d377dbcac3ae26794/README.md (visited on 03/16/2018).

[2] URL: https://github.com/ethereum/wiki/wiki/Whisper (visited on 03/16/2018).

[3] URL: http://ethdocs.org/en/latest/introduction/what-is-ethereum.html (visited on 03/16/2018).

[4] URL: https://bitcoin.org/bitcoin.pdf (visited on 03/16/2018).

Appendix

Project Definition

Subject

When the Internet was in its infancy, if you wanted to use it for a specific application, you might have written a protocol. That way, anybody who wanted use the Internet for that purpose would have a common way of doing it - and if a new person came along and wanted to join in, they could just write their software to conform with the standard.

In the past 15-or-so years however, the landscape has changed. Companies now favour their own proprietary systems, where they can have complete control, and ultimately gain the most profits. Specifically companies such as Uber have taken an industry which was once fairly well distributed, and put the control in their own hands - they decide who can be a driver, they manage the fares, and how much they pay their drivers.

But recent developments with distributed networks threaten to disrupt this comfy business model. Technologies such as Ethereum allow "trustless" applications, where activities of a single node are verified by the entire network. It's an area which is yet to be explored to its full potential, but all the features needed to be able to implement feature-rich apps are there. The logic of applications running on such a system has to be rethought, but with Uber as an example, there would be no central authority to take a cut of profits. The entire system would be self-regulating.

Deliverable

A ride-sharing webapp accessible with an Ethereum network-enabled browser, designed in such a way that no single entity has control over the running of the system. Drivers will be able to advertise their location (published to blockchain), and riders will be able to send job proposals (containing pickup and drop-off locations) to these drivers on a peer-to-peer basis. This protects the privacy of the rider by ensuring that only the chosen drivers are able to see the rider's location. When a driver initially advertises their location, they are required to provide a deposit to the network, which will be returned in completion of a trip. This gives the driver a stake in wanting to complete a journey, and should reduce spam on the network.

Drivers are then able to issue a response to a proposal, by either rejecting or quoting a price for the journey. This allows drivers to choose which journeys they take, and prevents drivers from having to travel a long distance to a pickup location, compared with if the allocation was done randomly. Should the rider choose to accept the quote, then both rider and driver form an agreement via a smart contract on the Ethereum network. This includes the passenger offering up the cost of the journey, plus an additional deposit equal to the amount the driver provided previously.

At this point, the fare for the journey, a deposit from the rider, and a deposit from the driver are all held by a smart contract. This acts as an incentive for the driver and rider to successfully complete the physical journey. When this is done, and both parties are in agreement that it is completed, then the deposits can be returned and the fare paid to the driver.

All monetary transactions will be executed with cryptocurrency on the Ethereum network, so as to minimise fees and prevent the transaction

from being intercepted by a third party.

As the vast majority of interactions between riders and drivers will be based on no existing knowledge of the other party, a reputation system will be used to form a layer of trust. Based on previous journeys, and the ratings given to both rider and driver on completion of each, future riders will be able to make informed decisions about which drivers to send job proposals. And in the same fashion, drivers will be able to decide which riders' proposals to accept.

Originality

Although ride-sharing apps aren't a new thing, nearly all existing solutions are controlled by a central authority who take a cut of the profits. This means users are at the mercy of the company when it comes to fares, and drivers must be approved, potentially opening the way for discrimination.

This project eliminates these problems by taking control away from any one part of the system. All transactions take place in a peer to peer nature, with the network being the only intermediary. This ensures that the two parties involved have full control over the process, whilst at the same time preventing one from cheating the other.

Timetable

The following proposed timetable will be used to track progress over the course of the project. The work is broken down into fortnightly blocks. Through the entirety, a project diary will be kept to keep track of key decisions. This is to be used as the basis for much of the final report.

Date	Planned Activity
02/10/17	Begin writing a formal project definition. Decide on project objectives, and have an idea of what features will be included. Which features would the system not work without.
16/10/17	Finish project definition. Begin mapping out interactions of users with the system and other users as a diagram. Create protocol documentation - similar to RFC. This is to be used to test against. **Project definition due 20th October**
30/10/17	Start implementing said protocol, with aim of creating fully function implementation (not including user interface). Test against RFC-style document.
13/11/17	Finish initial protocol implementation.
27/11/17	Develop testing suite for protocol implementation.
11/12/17	Fix any issues with implementation, and complete testing. Begin TP1 progress report.
25/12/17	Continue TP1 progress report.
08/01/18	Exams scheduled in this period, therefore expecing a slow down in project work. **TP1 progress report due 19th January.**
22/01/18	Continuation of development based on progress report. Begin writing of final report.
05/02/18	Research into how to develop the user interface. Review of existing mobile Ethereum clients.
19/02/18	Development of user interface.
05/03/18	Addition of identified "stretch" features.
19/03/18	Finalising development and report writing.
02/04/18	Report writing.
16/04/18	Finalising report and considering how to present the project during demos. **Final submission due 27th April.**

Teaching Period 1 Progress Report

This is the teaching period 1 progress review for my final year project, referred to here on after as *Taxicoin*.

Current Progress

As of January 2018, significant progress has been made on the implementation. From a technical point of view, the core "must have" features are complete.

As a rider, the user is able to advertise their job to drivers on an individual basis. The intention is that eventually the advertising will be done automatically, to all available drivers which meet some criteria, e.g. minimum rating or maximum proximity from rider.

When accepting a journey with a specific driver, the rider must pay the fare for the journey up-front, as well as an additional deposit which ensures the rider has a stake in completing a journey without dispute.

At the end of a journey, the rider is able to rate the driver. The rating acts as the only form of reputation, and is currently a simple average of all ratings. Each rating is an integer between 1 and 5.

Drivers are able to advertise their location publicly as an indication that they are active and accepting job proposals. However, to do so, drivers must provide a deposit.

In the event that a driver receives a job proposal, they have the option to respond with a quote for the fare of the journey.

Significant Achievements

- Technical contract implementation is now in a working state.
- User interface with map, location search, and user geolocation is in a working state.
- Spoke at BrumJS November 2017 meetup on the subject of the Ethereum platform and its uses. Afterwards gathered informal feedback about the concept of Taxicoin.

Next Steps

In terms of the implementation itself, the user interface needs tidying up significantly. At the moment, the "user flow" is somewhat lacking, and not as fluid as it should be. This is the first priority.

The reputation is currently very simplified from what I had initially planned. I would like to develop it further, as it is an integral part of the application. I've recently read into how other Ethereum-based decentralised applications are managing their reputation systems, and will hopefully apply some of the ideas in Taxicoin.

I plan to write an "Introduction to Ethereum" section of the report, with a comprehensive explanation of how the platform functions, and why I have chosen to develop Taxicoin with it.

As discussed with Peter Lewis, a crucial part in proving the successful implementation of a complete protocol for Taxicoin will be to develop a comprehensive suite of tests. These will primarily test the functionality of the contract parts of the application, as this is where the protocol layer is implemented.

Hurdles

The Ethereum platform is still rapidly evolving. Indeed, even from when I began research into how I would develop this project, protocol specifications have been amended. As a result, I am having to keep an eye on developments with Ethereum while developing the application.

Traditional databases for storing data do not translate well to blockchain-based systems. Therefore I will need to research distributed databases, particularly for a more advanced reputation system. This is unexplored territory for me, so I am unsure what to expect.

Project Diary

3rd October 2017 talked about the fact that this project is relevant to the interests of the ALICE group. It is effectively a self-governing application. Was decided that the focus should be on compiling a list of "must have" features and then implementing them.

16th October 2017 was suggested to write a RFC-style protocol specification, to be used later to test against to determine if the implementation is correct.

13th November 2017 no huge amount of progress was reported due to other commitments. We revisited the idea of producing an RFC-style document, focusing on the IMAP protocol as an example.

4th December 2017 we discussed that including a network architecture diagram in the report would be a good idea of explaining how various parts of the project communicate with each other (e.g. front end talks to contract, different instances of front end talk to each other). At this point, a working implementation had been completed, therefore we began talking about how to write tests. It was decided that the contract should be tested directly with unit tests, and potentially integration testing performed on the Javascript abstraction layer and contract. We discussed that it would be good to get to the point where the application could be security audited.

Ethics form for student projects

SEAS group: Computer Science

Project title: Taxicoin

Supervisor name and email: Peter Lewis <p.lewis@aston.ac.uk>

Ethics questions

Please answer Yes or No to each of the following four questions:

1 - Does the project involve participants selected because of their links with the NHS/clinical practice or because of their professional roles within the NHS/clinical practice, or does the research take place within the NHS/clinical practice, or involve the use of video footage or other materials concerning patients involved in any kind of clinical practice? ***No***

2 - Does the project involve any i) clinical procedures or ii) physical intervention or iii) penetration of the participant's body or iv) prescription of compounds additional to normal diet or other dietary manipulation/supplementation or v) collection of bodily secretions or vi) involve human tissue which comes within the Human Tissue Act? (eg surgical operations; taking body samples including blood and DNA; exposure to ionizing or other radiation; exposure to sound light or radio waves; psychophysiological procedures such as fMRI, MEG, TMS, EEG, ECG, exercise and stress procedures; administration of any chemical substances)? ***No***

3 - Having reflected upon the ethical implications of the project and/or its potential findings, do you believe that that the research could be a matter of public controversy or have a negative impact on the reputation/standing of Aston University? ***No***

4 - Does the project involve interaction with or the observation of human beings, either directly or remotely (eg via CCTV or internet), including surveys, questionnaires, interviews, blogs, etc? *Answer "no" if you are only asking adults to rate or review a product that has no upsetting or controversial content, you are not requesting any personal information, and the adults are Aston employees, students, or your own friends.* ***No***

Student's signature: Scott Street

Supervisor's signature: _____

Please submit this form as part of your Term 1 Progress Report. If any of the answers are "yes", you will need to complete an online application for ethics approval, which can be found at https://www.ethics.aston.ac.uk *. You can log in with your usual Aston user name and password.*