University of Dublin

Trinity College

# An Exploration of Novel Trading and Arbitrage Methods within Decentralised Finance

Stephen Byrne

B.A.(Mod.) Computer Science and Business

Final Year Project April 2021

Supervisor: Dr Donal O'Mahony

School of Computer Science and Statistics O'Reilly Institute,

Trinity College, Dublin 2, Ireland

# Abstract

Decentralised Finance (*DeFi*) is an umbrella term for the parallel financial system currently being built, without banks, on blockchain technology (Chase, 2020). Blockchain started as a transaction medium with its first implementation being the Bitcoin network (Nakamoto, 2008), however now the entire financial system is being reimagined on blockchain technology. The total value locked (*TVL*) in decentralised finance protocols on the Ethereum blockchain network, which are essentially autonomous applications on the blockchain networks, has grown from approximately $600m in January 2020 to approximately $25 billion in January 2021 (Werner et al., 2021). With large volumes of cryptocurrency transactions being processed daily, new opportunities present themselves for trading and arbitrage.

The goal of the project will be to explore decentralised finance by implementing and evaluating novel trading and potential arbitrage methods on the Ethereum network. This will involve augmenting methods from traditional centralised finance (*TradFi*) and foreign exchange trading (*Forex*), along with exploring new opportunities presented due to the technical architecture of decentralised finance being built on blockchain technology.

The author will devise trading methods that are viable for decentralised markets, examine the implications of such strategies, and build a proof-of-concept system implementing such methods for evaluation.

# Acknowledgements

First and foremost, I would like to thank my supervisor Dr Donal O'Mahony for the opportunity to research in the fascinating area of decentralised finance, and for his continued support, guidance and patience.

Secondly, I would like to express considerable gratitude to Corbin Page, who has been a source of both inspiration and guidance, as I have journeyed through the decentralised finance landscape.

Finally, I would like to thank my friends and family for their support during the tumultuous period of the COVID-19 pandemic during which this work was completed.

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1 | Introduction

## 1.1 Motivation

In 2017 Bitcoin garnered global media attention due to a large increase in its price. The author was an early advocate of the technology and was captivated by blockchain and its potential. In 2020, blockchain technology and its currencies (*cryptocurrencies*) and assets (*cryptoassets*) surged again in volume and price. Innovation occurred rapidly and new types of applications built on decentralised technology were created. One such example is the emergence of decentralised finance. The author previously worked in software engineering at a financial services company and became interested in this newly emerging form of financial technology. The author's interest in trading and arbitrage stems from a popular book which is based on real events titled "Flashboys" by Michael Lewis (2014). This book tells the journey of the protagonist who uncovers large scale corruption caused by unethical trading practices, coupled with high speed algorithmically automated arbitrage trading known as *High-Frequency Trading* (HFT). The author wished to investigate existing and potential novel trading strategies within the emerging field of decentralised finance, along with their implications and wider industry practices.

## 1.2 Approach

The approach of the author will be to first explain the underlying technologies that support decentralised finance such as blockchain technology, followed by trading and arbitrage, decentralised finance architecture and finally trading methods within decentralised finance. Traditional methods will then be tested in a decentralised finance context to examine if they are viable, followed by testing novel methods of trading which are made possible due to

innovations in decentralised finance. The author will also explain the drawbacks, investigate potential problems and come to a conclusion about the effectiveness and future of trading and arbitrage within decentralised finance.

## 1.3 Objectives

The overall objective of this project was firstly to master the elements of decentralised finance technology. This included investigating trading methods in decentralised finance, testing traditional and emerging trading methods, and examining any problems discovered through exploring these methods. To meet these objectives, sub-objectives were defined and detailed below:

The development of:

- A proof-of-concept trading bot to examine traditional financial trading and arbitrage methods within a decentralised finance context.
- A proof-of-concept trading bot to examine new emerging decentralised finance trading and arbitrage methods.
- Smart contracts supporting the above implementations.

The evaluation of:

- The viability of the implemented trading methods within decentralised finance.
- The wider impacts and implications of trading and arbitrage.
- The potential problems arising from such methods.

## 1.4 Report Structure

**Chapter 2** begins with the origin of blockchain technology and takes the reader through a journey of developments over the previous 10 years that have led to the emergence of decentralised finance as it stands today. Literature detailing the architecture of the decentralised finance ecosystem is described, from the base architecture of decentralised finance to the inner workings of the emerging exchanges and lending protocols, and their relationship to the trading strategies implemented. A look into the near future of exciting upgrades and their impact on the implementation is also considered.

**Chapter 3** details the functional and non-functional requirements, and a high-level system design that is described concerning the architecture of the blockchain network used in the implementation. Limitations of the chosen approach are considered, and an improved system design is proposed if the trading methods were to be implemented optimally.

**Chapter 4** commences the technical implementation of the proposed Proof-Of-Concept systems for the trading methods which were implemented. The architecture of the individual systems is described in detail, along with the technologies used and the rationale behind any choices made. The results of the trading strategies implemented are also evaluated.

**Chapter 5** provides a holistic evaluation of the trading strategies implemented, details the development and testing methodology and evaluates the fulfilment of requirements. Difficulties encountered by the author are also discussed and actions that were taken to overcome these difficulties.

**Chapter 6** concludes the report. The author summarises findings and gives personal thoughts on the state of decentralised finance ecosystem followed by concluding with a recommendation for future work influenced by his learnings.

## 1.5 Glossary

- **Fiat Currency:** Government-issued currency not backed by any commodity.
- **Cryptocurrency:** A digital asset designed to work as a medium of exchange implementing cryptography methods to secure the system, built on blockchain technology (Cryptocurrency, 2021).
- **Cryptoasset:** A digital asset representing a share of some pool or token, also functions as a medium of exchange and functions like a cryptocurrency but may implement additional features.
- **Token:** Interchangeable for cryptoasset or cryptocurrency.
- **Tokenomics:** Economics of a token.
- **Burn:** Destroy.
- **Governance Token:** A token that can be used to vote on decisions for decentralised protocols, governance tokens are a type of cryptoasset.
- **Stable coin:** A class of cryptocurrency that attempts to offer price stability and is usually backed by a reserve asset. An example being USDC which is backed by the dollar (Stablecoin, 2021).
- **On-chain:** Referring to data stored on the blockchain.
- **Off-chain:** Referring to data stored off the blockchain.
- **Bitcoin:** A blockchain network.
- **Ethereum:** A blockchain network.
- **Ether:** The cryptocurrency which powers the Ethereum blockchain.
- **Mainnet:** A reference to Ethereum's main network
- **Kovan:** A test network for Ethereum

# Chapter 2 | Background

## 2.1 Blockchain

### 2.1.1 Overview

Satoshi Nakamoto, a name presumed pseudonymous, is the creator/creators of Bitcoin, the first application of blockchain technology. Bitcoin was described by Nakamoto as a "purely peer-to-peer version of electronic cash" (Nakamoto, 2008). The white paper (Nakamoto, 2008) outlined the solution to solve the problem of 'double spending' with electronic currency by using a decentralised, peer-to-peer approach to verify and track transactions, without the transaction having to go through a traditional financial institution (Nakamoto, 2008).

Soon after Bitcoins creation the words blockchain and bitcoin became inextricably linked. However, the use of blockchain enables more than just a transfer of bitcoin between two parties. Numerous other blockchains have been developed along with standardised protocols for interacting with elements within the network. Ethereum is one such open source blockchain that features smart contract functionality and will be used in this work. A blockchain can be described as a public database that is updated and shared across many computers in a network (Nie et al., n.d.). In more technical terms, a blockchain serves as an immutable ledger that allows transactions to take place in a decentralised manner (Zheng et al., 2017). The "Block" refers to the data and state stored in sequential batches, and the "Chain" refers to each block cryptographically referencing its parent block's data; therefore, history can't be changed without changing all subsequent blocks which would require approval, or consensus, of the network (Nie et al., n.d.). An example of this architecture can be seen below (Figure 2.1), resembling a linked list-style data structure.

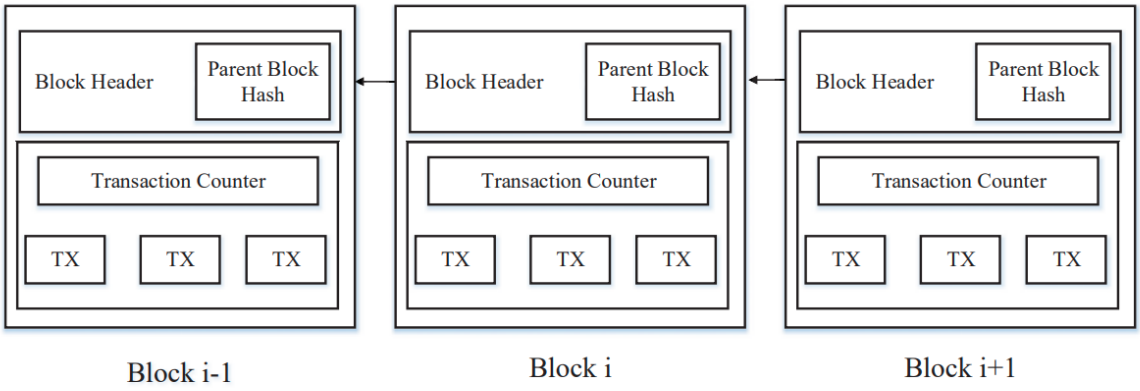


*Figure 2.1: Blockchain Architecture (Zheng et al., 2017)*

### 2.1.2 Characteristics and Features

#### Cryptography

Cryptography has been a key technology in the financial world for decades for securely storing and transmitting data, with uses in payment networks, ATM security, smart cards and online banking (Cachin, 2017) and involves applying mathematical functions to pieces of data to guarantee their security (How Does Cryptography Protect The Blockchain?, 2017). This usually involves using a hash function, which is a function that processes data of any size and converts it into a unique cypher-text of a specific length (Crane, 2021). Cryptography is generally divided into two categories, private key cryptography and public key cryptography. Central to Bitcoin and other blockchain networks like Ethereum is public key cryptography.

**Public Key Cryptography**

To explain public key cryptography an example from Bouidani (2015) will be given. Alice and Bob are two parties participating in communication as seen in Figure 2.2. Bob has a public and private key pair, the public key, $p$, can be shared over the network. Alice encrypts the message she wants to send, $m$, using Bob's public key and the encryption transformation $E_P$: $E_P(m) = c$. When Bob receives this encrypted message in the form of the cypher-text $c$, he must apply the inverse decryption transformation using his private key $v$: $D_v(c) = m$.

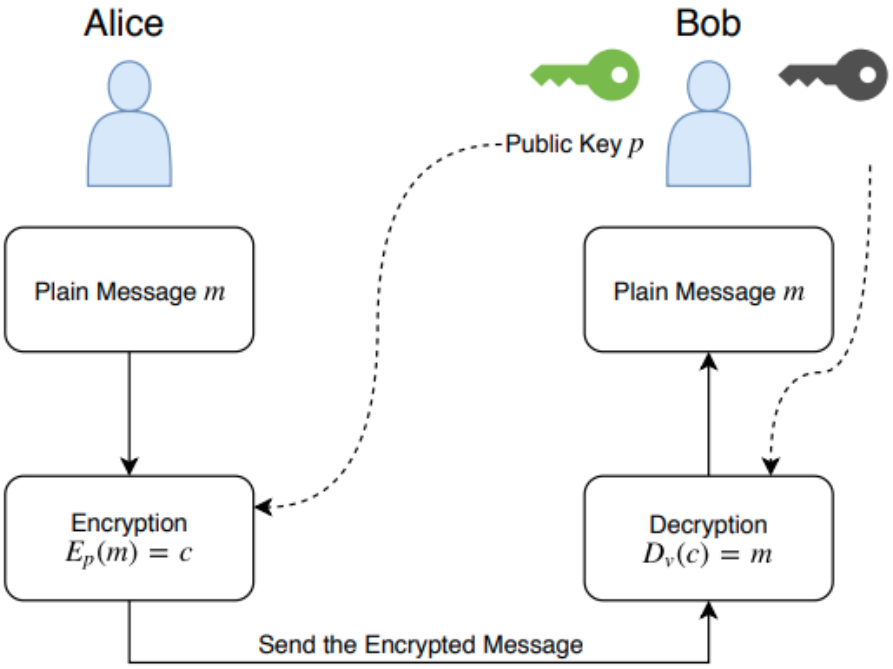


*Figure 2.2: Public Key Cryptography (Bouidani, 2015)*

**Distributed & Decentralised**

A blockchain can be described as a public database that is updated and shared across many computers in a network (Nie et al., n.d.). Distributed refers to the shared state of the network across nodes and a ledger is a collection of accounts. Therefore, a blockchain can be thought of as an immutable ledger that allows transactions to take place in a decentralised manner (Zheng et al., 2017) through shared technological infrastructure and protocols (Frankenfield, 2018). The networks are decentralised meaning there is no centralised party controlling the network. The concept of a distributed ledger is not limited to blockchain, and many organisations maintain data at different locations (Frankenfield, 2018) and use a centralised system as a single source of truth to keep the records state in line. Blockchain differs by removing the single source of truth and maintaining the network state through consensus mechanisms after a transaction takes place on the network.

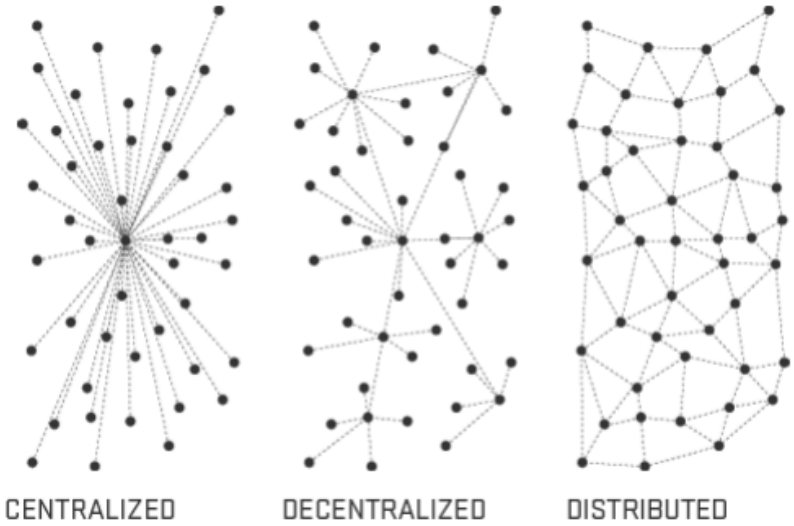


*Figure 2.3:Comparison of Network Architectures (Swanson, 2015)*

### 2.1.3 Transactions and Contracts

#### Smart Contracts

The term smart contract was introduced in 1994 (Szabo, 1994) as an electronic "transaction protocol that executes the terms of a contract" however with the rapid growth of Ethereum, the first blockchain implementation of smart contracts, the definition of a smart contract has evolved. Smart contracts are neither smart nor contracts in a legal sense (Zetzsche et al., 2020), they are self-executing software protocols that reflect some agreement between two parties. The conditions of the contract are written into lines of code and are executed upon certain conditions being met. The two parties engaging with the contract may be human or two other smart contracts, which in turn could be owned by other smart contracts or humans. The smart contract simply permits the execution of code upon agreed terms, without the need for an external enforcement mechanism (Zetzsche et al., 2020). Smart contracts have the ability to transform processes and industries such as supply chain management, agriculture, real estate and finance. (Hewa et al., 2021). The core of most decentralised finance protocols revolves around processes driven by smart contract interaction, known as transactions. Smart contracts are also particularly interesting concerning trading and arbitrage as they allow atomic processing of batches of trading actions by calling other contracts methods from within the starting contract (Daian et al., 2019). This allows the bundling of several actions to be included in a singular smart contract and executed together within one transaction.

#### Transactions

As seen in Figure 2.1, each block consists of several transactions. Transactions change the state of the blockchain network and must be verified by a consensus mechanism before being

included in the latest block. This is a process known as validation, and after completion, the network state is updated, and the transaction included on the blockchain on every node.

### Hash Functions

Bitcoin and Ethereum both utilise a hash function to generate a unique output for each block that functions like a digital fingerprint (Giesen, 2020). An algorithm is used to take the contents of a block and hash it to generate this unique fixed-size output. The hash of the new block created uses both the hash of transactions it contains and the hash of the previous block. As a result of this process, each block is linked to the previous block through its hash (Bouidani, 2015) adding immutable history to the blockchain. This is visualised below in Figure 2.4.

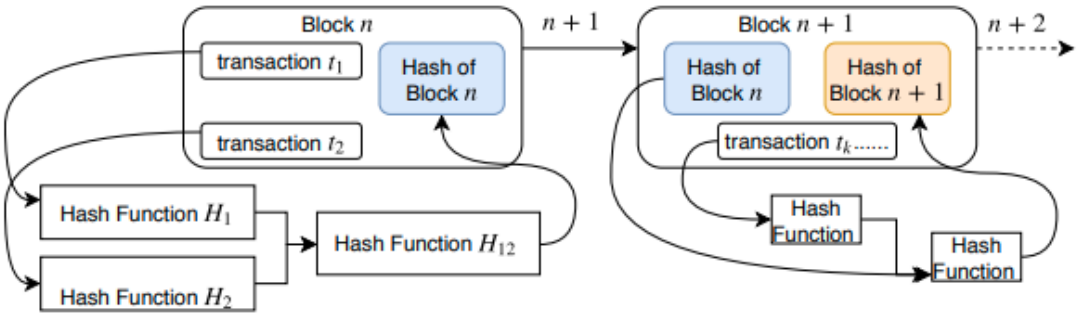


*Figure 2.4: Hash Functions in Blockchain (Bouidani, 2015)*

### Merkle Trees

Merkle Trees were first proposed by Ralph Merkle (1988). They are a tree structure where every leaf node is labelled with the cryptographic hash of a data block, and every non-leaf is labelled with the hash of the labels of the child nodes (Merkle tree, n.d.). The root node is therefore a combination of previous hashes and is known as the Merkle root. To prevent blocks

from storing every hash within a blockchain network, a Merkle tree is used and the Merkle root included in the latest block (Nakamoto, 2008). Ethereum uses a modified version of this known as a Patricia Tree (Cordell et al., n.d.).

### 2.1.4 Consensus

In traditional network architecture, a centralised system, a single source of truth is decided upon by a single controlling entity for the state of the system (Muzzy, 2020). As blockchain technology functions in a distributed manner with a network consisting of individual nodes and no single controlling entity, there must be a system for the nodes to reach an agreement on the current system state. This is achieved through consensus mechanisms which are (Wackerow, Richards and Cordell, n.d.) implemented with cryptography, hence where the term "crypto" originates from when describing blockchain-based currency. The two most popular consensus mechanisms are called Proof-of-Work (PoW) and Proof-of-Stake (PoS).

**Proof-Of-Work *(PoW)***

Bitcoin and Ethereum both launched with a proof-of-work consensus mechanism, which has been extremely resilient and secure to date. The concept of Proof-of-Work originates from a paper written by Cynthia Dwork and Moni Naor (1993) with its first popular implementation being by Adam Back (1997) in the HashCash system as a solution to limit email spam and denial-of-service attacks. To send an email within the HashCash system, a cryptographic puzzle must first be solved which adds a small computational cost to sending an email, a replication of the concept of a physical stamp in the traditional mailing system. The solution was to construct the information contents of a stamp including recipient, date and version and to

append a random string, which is a number that is only used once (nonce), to this before the result was hashed. The first 20 bits were required to be 0 and if this was not met, the nonce was incremented and it was attempted to calculate a suitable hash again. The leading 0's imposes a level of difficulty to guessing the correct hash (Frankenfield and Anderson, 2021).

Blockchains like Ethereum and Bitcoin use this system today, replacing the hashed data with the hash of all transaction data in the block. This validation method became known as "mining" where the "miners" are the individuals or entities that run nodes to maintain the network and solve these problems to validate the transactions. The miner who solves the problem earns a reward for their work, often in the form of the cryptocurrency the network is supported by and shares the newly validated block with the rest of the network (Cordell and Richards, n.d.). The leading number of 0's required for the hash is also adjusted, the difficulty (Frankenfield and Anderson, 2021), to keep the time of solving the blocks constant as more miners join the network which in turn increases the rate at which the hashes are solved, known as the hash rate. Proof-Of-Work eliminates the need for a central decision maker for consensus of the network (Frankenfield and Anderson, 2021). In blockchain networks that run a Proof-of-Work system, the longest chain of blocks is considered the chain with the greatest proof-of-work invested into it (Nakamoto, 2008). If the majority of the mining power is controlled by honest parties, the honest chain grows the fastest and outcompetes dishonest chains (Nakamoto, 2008). To modify a past block an attacker would need to redo the proof-of-work of all previous blocks and then surpass the greatest honest chain. To break this system miners would need to control over 51% of the mining power, this is known as a 51% attack (Frankenfield and Anderson, 2021).

The problems of proof-of-work are a result of the method of expending energy to solve these cryptographic problems to validate the transactions. As the network scales, the difficulty is

adjusted and the energy required increases as more miners join the network (Frankenfield and Anderson, 2021). The Bitcoin network, for example, has increased in energy consumption as it has grown, and now consumes more energy than the entire country of Argentina (Criddle, 2021). This is problematic, and due to this, alternate consensus mechanisms have been proposed. One is proof-of-stake which Ethereum is transitioning to using.

**Proof-Of-Stake *(PoS)***

Ethereum launched running a proof-of-work consensus model, however today it has begun to transition to a proof-of-stake consensus model to combat the problems of proof-of-work. Staking can be broadly defined as the act of locking up a cryptocurrency to acquire some form of return, in the context of proof of stake, the return is made by validating new blocks to add to the network (What Is Staking? | Binance Academy, 2021). Proof-of-stake uses validators instead of miners. These validators stake some capital, usually in the form of the cryptocurrency supported by the network, and validators are chosen at random to create new blocks. This eliminates the heavy computational work from proof-of-work. To ensure security, if a validator acts with malicious intent and tries to compromise the system, the validator loses its stake. Therefore, for proof-of-stake to succeed the amount of capital staked must be more than the reward for generating the new block or the penalties for acting maliciously. This means if a validator acts with malicious intent they will lose more than they gain. There are also partial penalties for the node being offline (Wackerow, Cordell, Stockinger and Richards, n.d.).

## 2.2 Ethereum

### Overview

The Ethereum blockchain was first proposed by Vitalik Buterin (2013), with the goal of allowing anyone to write smart contracts on blockchain technology. Due to its implementation of smart contracts, which support a Turing complete language, it has more utility than Bitcoin. Ethereum surpassed Bitcoin to become the most widely used blockchain in 2020 (Madeira, 2020). Ethereum is the name of the blockchain network, while Ether is the currency that powers the network. As mentioned previously, when Ethereum launched in 2015 it used a *PoW* model similar to that of the Bitcoin network. Today, it is transitioning to a *PoS* model to improve efficiency, this is being implemented over numerous years and the first "node" launched successfully on December 1st, 2020 (Young, 2020). While Bitcoin is considered a distributed ledger of transactions, as a result of the smart contract functionality the Ethereum network functions more like a distributed state machine (Cordell et al., n.d.). The network has a set state which can change from block to block based on the smart contracts and interactions with them.

### 2.2.1 Token Types

#### Ether

Ether is the cryptocurrency that supports the Ethereum network. Miners and validators earn Ether as a reward for processing transactions. Modifying the state of the Ethereum network incurs a cost in Ether, such as by transacting or deploying a contract. The Ethereum network usually stores token balances in Ethers smallest unit, Wei. The value conversions can be seen below in Figure 2.5. As we can see, 1 Eth is equivalent to $1 * 10^{18}$ Wei. This is useful for calculations within trading discussed later in this work.

| Unit | Wei Value | Wei |
|---|---|---|
| wei | 1 wei | 1 |
| Kwei (babbage) | 1e3 wei | 1,000 |
| Mwei (lovelace) | 1e6 wei | 1,000,000 |
| Gwei (shannon) | 1e9 wei | 1,000,000,000 |
| microether (szabo) | 1e12 wei | 1,000,000,000,000 |
| milliether (finney) | 1e15 wei | 1,000,000,000,000,000 |
| ether | 1e18 wei | 1,000,000,000,000,000,000 |

*Figure 2.5: Unit Conversion for Ether (What is Gwei? - ETH Gas Station, 2019)*

**ERC-20 - Fungible Tokens**

ERC-20 (Ethereum Request for Comments 20) was a proposal by Vogelsteller and Vitalik (2015) to introduce a token standard for fungible tokens (Cordell et al., n.d.). The tokens can be used to represent nearly anything, from a claim on a fiat currency like USD, an ounce of gold or reputation points in an online platform (Cordell et al., n.d.). Fungible means that each token is equivalent to another token of the same type. For a smart contract to be considered an ERC-20, it must implement certain methods as outlined in the standard and will be responsible to keep track of the created tokens (Cordell et al., n.d.).

**ERC-20 Standard Methods**

```
function name() public view returns (string)

function symbol() public view returns (string)

function decimals() public view returns (uint8)

function totalSupply() public view returns (uint256)

function balanceOf(address _owner) public view returns (uint256 balance)

function transfer(address _to, uint256 _value) public returns (bool success)

function transferFrom(address _from, address _to, uint256 _value) public returns
(bool success)

function approve(address _spender, uint256 _value) public returns (bool success)

function allowance(address _owner, address _spender) public view returns
(uint256 remaining)
```

**ERC-20 Standard Events**

```
event Transfer(address indexed _from, address indexed _to, uint256 _value)

event Approval(address indexed _owner, address indexed _spender, uint256 _value)
```

These tokens are often referred to as “cryptoassets” with one popular example being protocol governance tokens which are often compared to shares in a company in traditional finance and offer some governance features such as voting on the future development of the applications. All tokens have a value that is set by the market, some of which are influenced by a peg (such as a peg to the dollar). A token is worth what you can trade or swap it for. The underlying asset being represented is not particularly important for this work, however, holding volatile tokens which may fall in value is likely not a good idea.

### 2.2.2 Account Types

An Ethereum account is defined as an entity with an Ether (ETH) balance that can send transactions on Ethereum. There are two types of accounts, externally owned accounts and contract accounts (Tesař et al., n.d.). Both types of accounts can send, receive and hold ETH and tokens and can interact with contracts that are deployed on the Ethereum network. Both types of accounts also have some common fields such as the nonce, which is a counter that indicates the number of transactions sent from an account, ensuring transactions are only processed once, and the balance, which is the amount of Ether owned by the account, denominated in the unit of Wei.

**Externally-owned Accounts (EOAs)**

Externally-owned accounts are controlled by the private key holder(s). They have no cost to create and any user on the network can generate accounts. They do not need to be "opened" like bank accounts. Each account has a cryptographic pair of keys, the public and private keys. They implement public-key cryptography to sign transactions, as described previously. This allows custody over the funds in the account to be controlled by the private key holder and security as the funds are stored on Ethereum's ledger (Tesař et al., n.d.).

**Contracts**

A contract, or smart contract, is a program that runs on Ethereum. These have been previously explained functionality-wise, but they also function as a form of account. They have the same 4 common shared features as externally-owned accounts. Contracts, unlike externally owned accounts, have a cost to create as they use storage space on the Ethereum network. They are an instance of a particular contract object that has been instantiated on the blockchain with methods that can be invoked. Their creation, by being "deployed", changes the network state as they are added to the blockchain. Smart contract functions are triggered in response to receiving an input and can execute any piece of logic encoded into the contract such as transferring tokens or interacting with other contracts (Tesař et al., n.d.), this happens within a transaction.

**Wallets**

A wallet is a piece of software used to manage accounts. Wallets securely store the keys associated with the accounts and are used to generate and sign transactions. These are not externally-owned accounts, but allow the management of externally-owned accounts. There are many different wallet software's, one of the most popular is called MetaMask. These often provide a friendly user interface for managing transactions. They will not be used in this work as the implementation will use code to control an account and execute transactions.

### 2.2.3 Transactions in Ethereum

A transaction in Ethereum is defined as a cryptographically signed instruction from an account (Rattanaveerachanon et al., n.d.). Transactions are initiated by a wallet and signed by an external-owned account (EOA) and change the state of the blockchain upon completion.

Transactions have a fee which is variable. Each transaction submitted has a gas price, which is the price paid per unit of gas, and a gas limit which is the maximum amount of gas the user will allow the transaction to use. The total transaction "fee" is the amount of gas used multiplied by the cost per unit gas the initiator of the transaction is willing to pay. The amount of gas used by the transaction varies based on the number of state changes taking place on the network. Any gas not used in a transaction is refunded to the sender upon completion (Rattanaveerachanon et al., n.d.). The gas price varies based on how fast the sender would like the transaction mined, as miners will prioritise transactions with higher gas prices as they keep the fee (Rattanaveerachanon et al., n.d.). A simple transaction may send Ether from one account to another requiring a small amount of gas, while a more complex transaction may transfer numerous types of ERC-20 tokens across several smart contracts which would require more gas.

A submitted transaction must include the following information:

1. The recipient's address.
2. The signature of the sender is generated when the sender signs the transaction with their private key.
3. The value of Ether to transfer.
4. An optional field of other data (such as token transfers).
5. The gas limit is the maximum amount of gas that the transaction will consume.
6. The gas price is the fee the sender would like to pay per unit of gas (Rattanaveerachanon et al., n.d.).

Transactions can exist in one of three states, *unconfirmed* and not validated, *confirmed* and considered executed, or *rejected* as invalid by the network of peers (Daian et al., 2019).

**The Lifecycle of a Transaction**

1. First a transaction is created by a user and a transaction hash is calculated.
2. The transaction is then broadcast to the network and enters the Mempool in every mining node, where it waits for a miner to pick it to be included in a block. Each miner has their own Mempool within their node. This is where miners prioritise higher gas fees, resulting in them having quicker transaction times (C., 2020). Here the transaction is considered *unconfirmed* or pending (Docs.dfuse.io, 2021).
3. After the transaction is mined and verified, the transaction state is then considered to be "in the block", however it is not yet *confirmed* (Docs.dfuse.io, 2021).
4. When the subsequent child block is mined, the transaction is considered *confirmed*. This dependence on the subsequent child block being mined is due to certain blocks being reversed by the network, and therefore the transactions reverting to the *pending* state (Docs.dfuse.io, 2021).

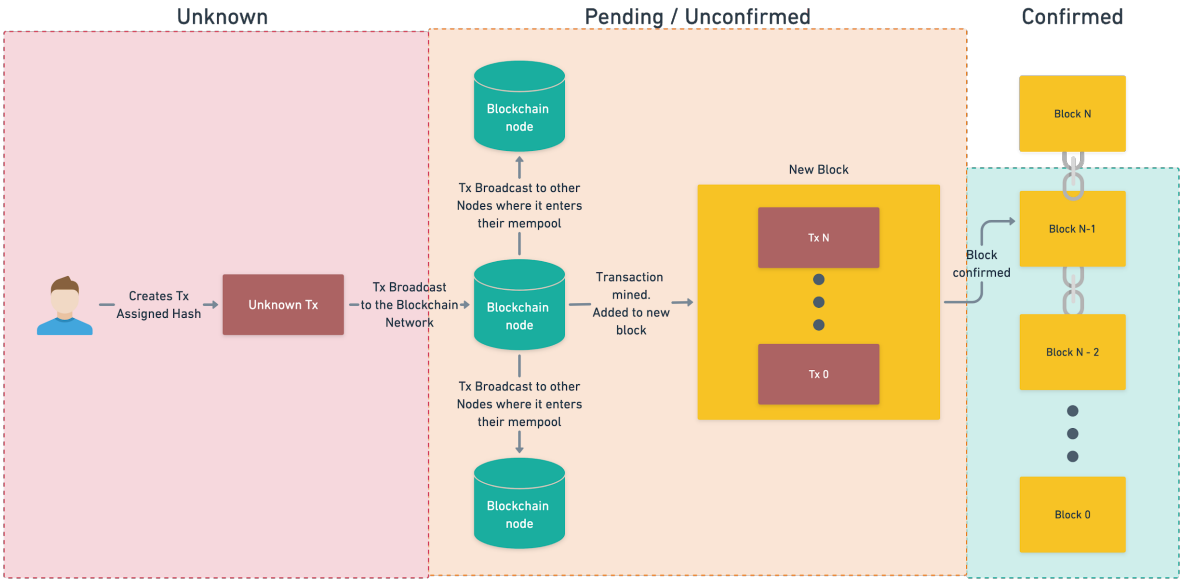


*Figure 2.6: Life Cycle of an Ethereum Transaction*

There are also alternatives to this flow displayed in Figure 2.6. Under normal circumstances the transactions completion causes termination however if a transaction fails due to any errors within the execution of a smart contract or requirements which are not met, the transaction will be *rejected*. In the author's implementation, this will be taken advantage of as the transaction will be rejected if it is not profitable, however some Ether will be lost as a result of the gas fee on failed transactions.

The transaction does also not specifically have to follow this flow of *unconfirmed* to *confirmed.* Other states have been defined by the Ethereum community, such as *unknown.* A transaction may move directly from the *unknown* to a *confirmed* state (Docs.dfuse.io, 2021). This is often the case when the party submitting the transaction has access to a private node or private group of nodes that do not broadcast their transactions to the rest of the network and subsequently mine the block themselves. This will be considered later in this piece of work.

### 2.2.4 The Mempool

The *Mempool*, also referred to as the "memory pool", "transaction pool" or "transaction queue" is the set of in-memory data structures inside an Ethereum node, that stores potential transactions before mining which adds them to the blockchain (Blocknative.com, 2020). It acts as a cache that grows and shrinks based on the number of transactions waiting to be processed (Ahmad, Saad, Bassiouni and Mohaisen, 2018). A high-level visualisation of how the Mempool, transactions and the blockchain systems operate in tandem can be seen in Figure 2.7.

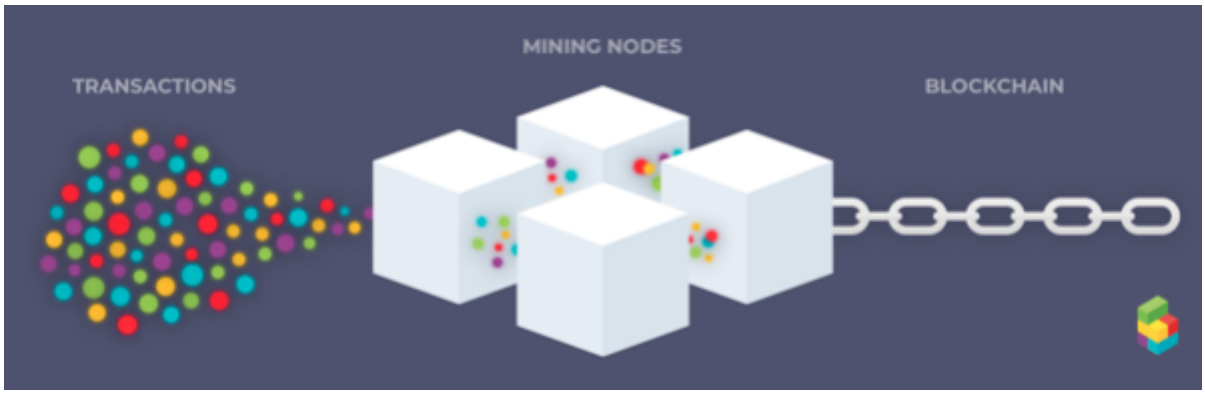


*Figure 2.7: Transactions Being Added to the Mempool before being Mined (Blocknative.com, 2020)*

Each node in the network, which is controlled by a miner, has its own Mempool. Often when referring to "the Mempool" the collection of all of these individual Mempools is being referred to as a whole. When a transaction is submitted to the network, nodes usually relay the transaction across the network to other miners (Deneuville, 2021). Because of this, the nodes are not always perfectly in sync due to network communication being unreliable and nodes having different rules about which transactions they accept, often preferring higher gas prices to maximise profit from mining (Blocknative.com, 2020). The Mempool is particularly interesting concerning arbitrage as it allows a trader to see unprocessed transactions. This will be described later in the decentralised exchange arbitrage section of this work.

### 2.2.6 Decentralised Applications

A decentralised application (dApp) is any application that runs on a P2P network or distributed system (Decentralized application, 2021). Decentralised Finance, which will be explained in detail later in the report, is a collection of such decentralised applications on blockchain which are interlinked to form a financial services system.

## 2.3 Decentralised Finance

### 2.3.1 Overview

Decentralised Finance (*DeFi*) is an umbrella term used for the parallel financial system being built with blockchain technology without banks (Chase, 2020). It differs from traditional banking as individuals generally remain in control of their funds and data and there are no borders or requirements to access bar an internet connection (Chase, 2020). Traditional finance *DeFi* equivalents can be seen below in Figure 2.8.

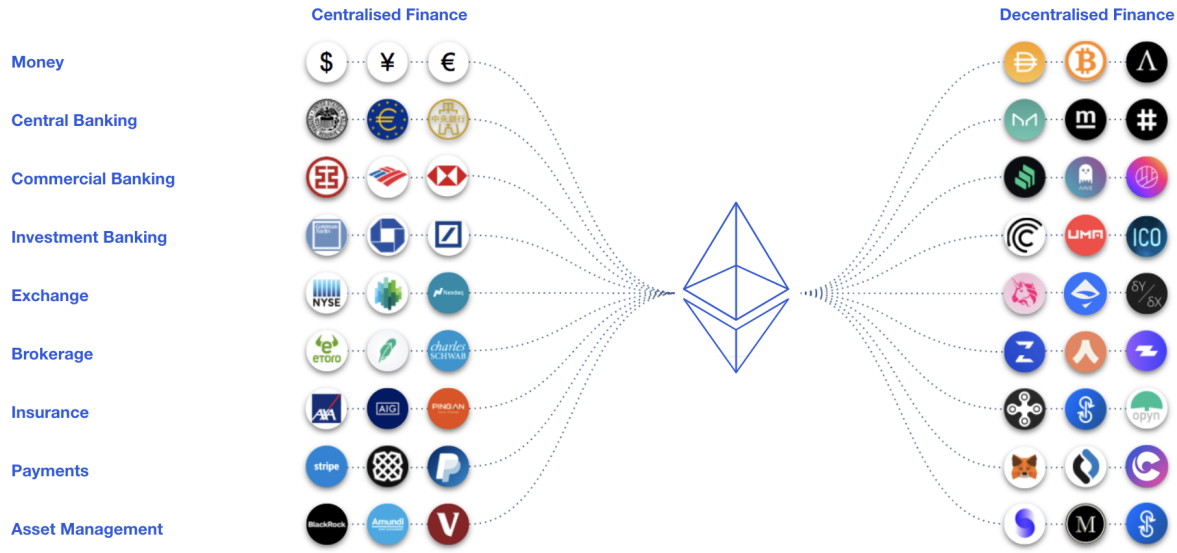


*Figure 2.8: DeFi Use Cases (Page et al., 2020)*

### 2.3.2 Key Characteristics

Decentralised finance has some key characteristics that are starkly different from traditional centralised finance. These characteristics are highlighted by Meegan and Koens (2021) and the characteristics which are most impactful on the implemented trading strategies are highlighted below.

**Composability**

Composability is ultimately what enables the implementation of the trading methods in this work and is arguably the defining property of Decentralised Finance (Meegan and Koens, 2021). As mentioned by Meegan and Koens (2021), Gudgeon et al. (2020) define composability within the context of decentralised finance as "the ability to build a complex, multi-component financial system on top of crypto-assets." Meegan and Koens (2021) also mention the common metaphor given in the decentralised finance community of "Money Lego" in which each "Lego" created can build on top of the other pieces of Lego in the system, where Lego represents groups of interacting smart contracts on the blockchain known as decentralised finance protocols. The components can be easily connected to (Popescu, 2020) by any party who follows the rules defined within the contracts to interact with them. This is a stark contrast to the proprietary nature of centralised payment systems (Meegan and Koens, 2021).

**Interoperability**

Schar (2020) defines two types of interoperability, functional interoperability and technical interoperability. Functional interoperability refers to services that can work together because they exist on the same platform, and technical interoperability referring to two different platforms which can work together. Decentralised finance enables both technical and functional

interoperability, although it is more enabling of the latter. Nearly every decentralised finance protocol is built with functional interoperability in mind. In the case of the implemented trading strategies, this will allow the borrowing of capital, and trading across numerous exchanges within a single transaction.

### 2.3.3 Architecture

Understanding the architecture of decentralised finance allows us to connect previously explained concepts into one system. Decentralised finance has a multi-layered architecture with its foundation being the Ethereum Network. Schär (2020) breaks this into five separate layers. The settlement layer (1) is the blockchain network, Ethereum, and its native asset Ether. This allows secure storing of ownership information and ensures state changes are accepted by the network through its consensus mechanism. The asset layer (2) is a collection of tokens issued on top of the settlement layer. On Ethereum, this consists of ERC-20 tokens which are fungible tokens and non-fungible ERC-721 tokens. Many decentralised protocols issue their own tokens, which exist on this layer. The protocol layer (3) consists of the backend layers of the decentralised applications which support the decentralised finance ecosystem, these are built as a collection of interlinked smart contracts. Some examples of these can be seen in Figure 2.9, for example, smart contract-based exchanges. These protocols can be accessed by users or other smart contracts and are highly interoperable. The application layer (4) is the user-facing applications that connect to protocols. These abstract away smart contract complexities and let users easily access protocol functionality, often through connecting their wallet in which funds are stored and used to sign transactions into the protocol. The aggregation layer (5) creates user-centric platforms connecting numerous applications and protocols. These further abstract away the complexities of transacting across multiple protocols, often simultaneously (Schär, 2020).

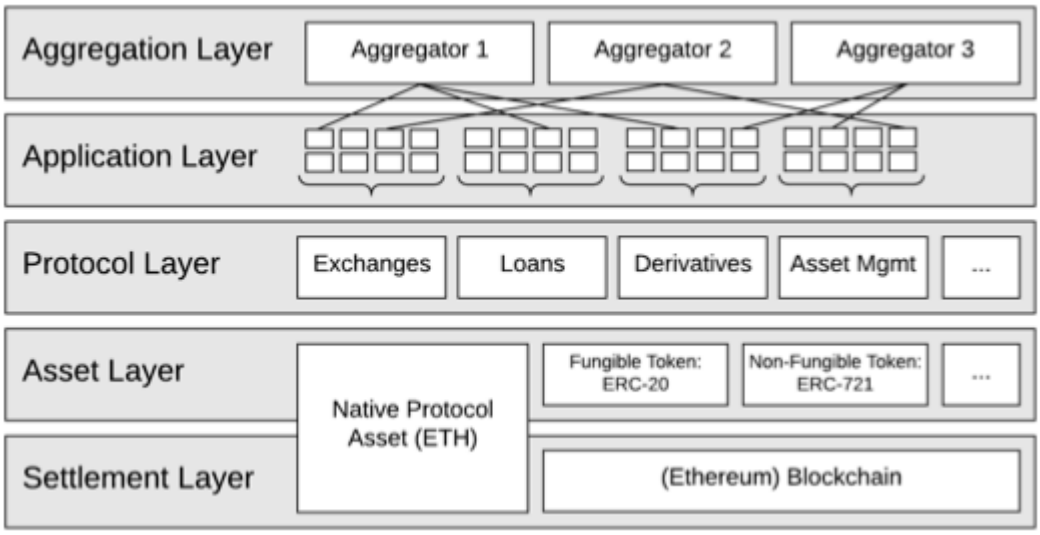


*Figure 2.9: Decentralised Finance Architecture (Schär, 2020)*

## 2.4 Exchanges

Similar to traditional financial markets, the trading venues for cryptocurrencies or tokens are called exchanges (Schueffe & Groeneweg, 2019). Exchanges can be split into two main categories, centralised exchanges (CEX) and decentralised exchanges (DEX).

### 2.4.1 Centralised Exchanges

Centralised Exchanges are known as such because they are controlled by a central party. They are efficient and the standard medium for trading traditional financial instruments in the traditional finance landscape. The most popular centralised exchange for trading cryptoassets is called Coinbase. Before decentralised finance, all trading and buying of cryptoassets were done on centralised exchanges. When considering decentralised finance, there is a movement to remove the need for a middleman, which the centralised exchange acts as in transactions.

This is where the clear disadvantage of centralised exchanges is exposed, to trade on a centralised exchange a user must deposit assets into the exchange effectively removing their direct access to the asset and relying on the honesty of the exchange operator to act according to their will. There is also the risk of the exchange being a single point of failure in the system (Schär, 2020). Severe high-profile failures have already occurred, with one of the most notable being the $40m stolen from a popular exchange called Binance (Russel, 2019). These systems usually have an order book system that matches buyers and sellers.

### 2.4.2 Decentralised Exchanges

Decentralised exchanges are applications built from numerous interacting smart contracts, known as protocols. Decentralised exchange protocols mitigate the risks of the central exchanges by removing trust as a requirement and being operational fully on-chain, meaning there is no single source of failure (Schär, 2020). There are also numerous types of decentralised exchanges and each function with different mechanisms. The pricing mechanisms implemented by the exchanges are particularly important concerning trading and arbitrage. A technical overview of some of the exchange types is described below.

**Order Book Exchanges**

An order book can be defined as a list of buy and sell orders for a specific financial instrument organised by price level (Order Book, 2021). Traditional finance stock market exchanges function with an order book, as do most centralised cryptoasset exchanges. There are two types of order book exchanges within the decentralised finance system.

On-chain Order Books

On-chain order books store all orders within smart contracts, visible to any user of the network. These are considered fully decentralised and require no third-party hosts or infrastructure. This is appealing from a centralisation standpoint as there is no single source of failure, however, due to the current speed of the blockchain networks and costs associated with writing data and transacting, there are disadvantages to this approach. Market volatility also requires frequent order cancellations due to changing prices which further congests the network leading to a cyclic problem of slowing speeds and higher costs (Schär, 2020).

Off-chain Order Books

Due to the disadvantages of fully on-chain systems, many protocols use an off-chain order book and choose to only use the blockchain as the settlement layer (Schär, 2020). For example, the so-called "0x" protocol (Warren and Badeali, 2017) uses this method. The order books are hosted and updated by third parties known as relayers (Schär, 2020). This method functions like a traditional order book exchange, where a "maker" sends a trade they would like to make which is pre-signed to the relayer who includes it in the order book. "Takers" can then query the order book and submit a signed transaction to complete the order to a smart contract which triggers the atomic exchange of the assets (Schär, 2020).

**Smart Contract Liquidity Pooling Systems**

Schär (2020) defines a liquidity pool as "a smart contract that holds at least two cryptoassets and allows trading through depositing a token of one type and thereby withdrawing tokens of the other type." These liquidity pools use a formula to determine the exchange rate, with the most widely used being the *constant product model*. These pools price assets purely on-chain due to this and rely on no external parties or price feeds. Every transaction affects the balance of tokens in the pool and therefore the exchange rate due to the constant product model, and it has no outside links to the wider market. Due to this, trading opportunities present themselves, and upon being acted upon by arbitrageurs the market is brought back into equilibrium. These are often referred to as automated market makers (AMM), one such example being the Uniswap exchange.

Constant Product Model

The constant product model can be expressed in its simplest form as $x * y = k$ where $x$ and $y$ represent the reserves of the tokens within the pool and $k$ represents some constant (Schär, 2020). Smart contracts which implement this model for trading are referred to as constant function market markers (Schar, 2020) with one of the earliest implementations being the Bancor exchange (Hertzog et al.,2017). As we can see from this formula, if we were to withdraw asset $x$ in this liquidity pool, we would need to supply asset $y$ to ensure the $x * y = k$ equation holds. This would reduce the reserves of token $x$ while increasing the reserves of token $y$. As Schär formally (2020) explains, when a trade is executed, the formula can be described as $(x + \Delta x) * (y + \Delta y) = k$. It can be shown from here that $\Delta y = \frac{k}{x + \Delta x} - y$, therefore $\Delta y$ will assume negative values when $\Delta x > 0$. In relation to a token swap through a pool with this implementation, this means supplying some amount of token $x$ to remove some

amount of token $y$ or vice versa. Under this model, the balance of the tokens in a liquidity pool can never be depleted as the token will get infinitely more expensive as the reserves approach 0 (Schär, 2020). This leads to differentiating between rate and return. The spot-price (rate) for a token can be calculated as $X/Y$ for any pair of tokens, however, this will not be the return. In traditional finance, the rate and return are equivalent. For example, if one goes to the bank and the exchange rate is £1.5/$ it means for every dollar one exchanges one will receive £1.5. If one were to exchange $2, one would receive £3. This is not the case for liquidity pools due to the constant product formula model. Returns are not linear.

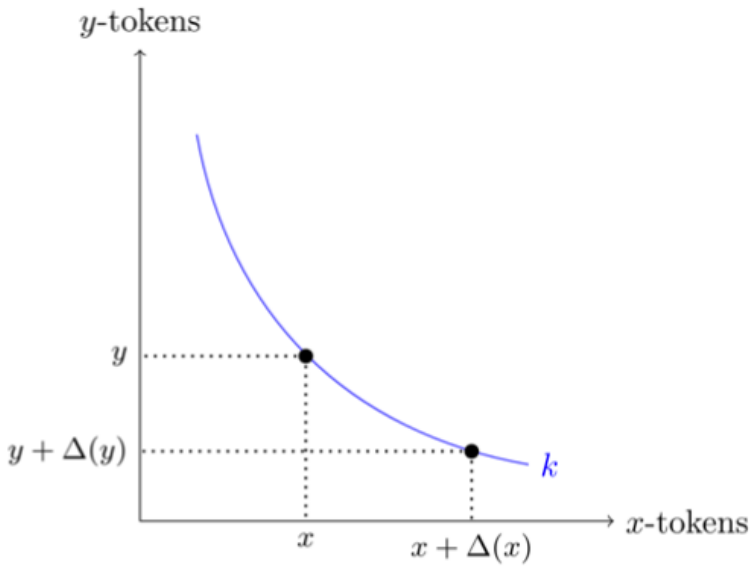


*Figure 2.10: Visualisation of Liquidity Pool CPM (Schär, 2020)*

When examining Uniswap (Adams, 2018), a decentralised exchange that implements a liquidity pooling system using the constant product model, the flow of how one executes a transaction to trade tokens, and the effect it will have on a given pool can be examined and is visualised below in Figure 2.11.

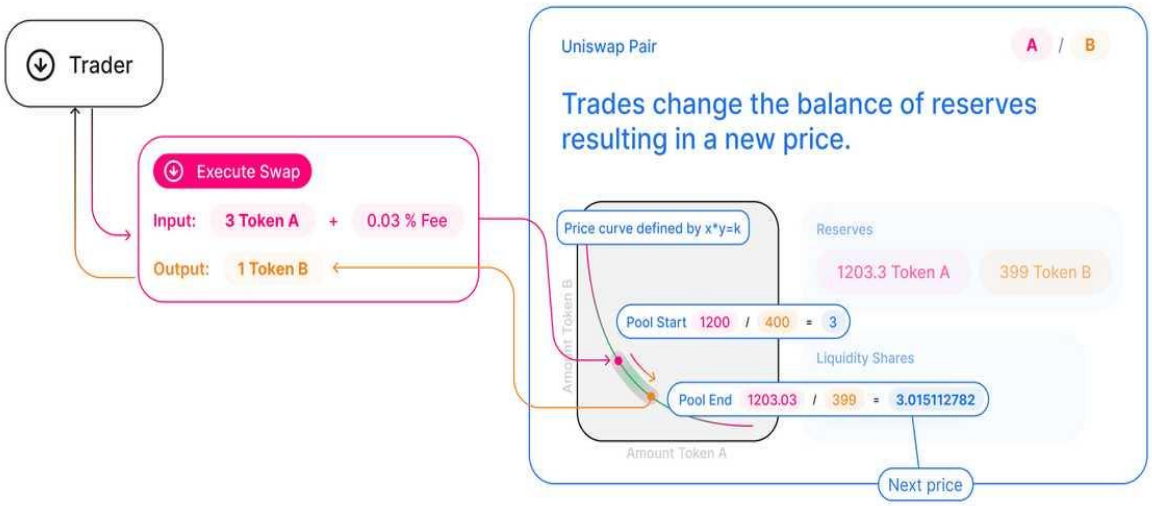


*Figure 2.11: Uniswap Transaction Example (Uniswap | How Uniswap works, 2021)*

As seen in Figure 2.11, the ratio of the tokens in the pool determines the spot exchange rate. A user can also define the amount of "slippage" they are comfortable with accepting, which is a percentage change in the submitted rate due to surrounding transactions by other users. This ensures there is not a substantial move outside of their price boundaries as a result of surrounding trades in the block or previous blocks.

The price impact of different trade sizes is visualised in Figure 2.12. On the left side of the figure, which is the Uniswap user interface, when swapping 10 Ether for Uni the price paid per Uni is approximately 0.014 Ether, however as seen on the right side of the figure when submitting a much larger trade of 10000 Ether, the price paid per Uni increases to approximately 0.043 Ether per Uni.

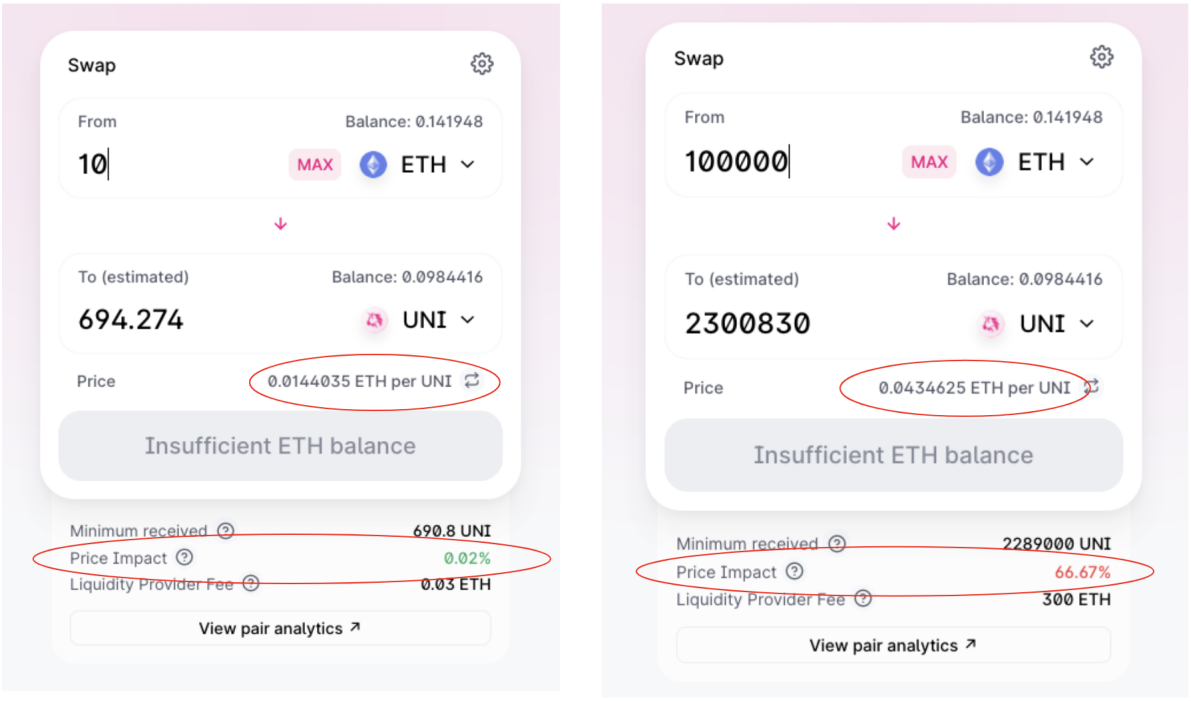


Figure 2.12: Price Impact of Different Trade Sizes on Uniswap

Uniswap also adds a small fee (0.3%) to all transactions which is added to the reserves in the pool. This has the effect of shifting $k$, as seen in Figure 2.13, thus increasing the total number of tokens in the pool. This functions as a pay-out to liquidity providers (LPs), the people who deposit their tokens in the smart contract to provide liquidity for others to swap through. When the liquidity providers (LPs) deposit the pair of tokens, they receive a new token representing their share of the pool. These new tokens may be redeemed at any point in time, at which point they receive a pay-out proportional to the amount of fees earned from their share of provided liquidity and their share of the pool's tokens are returned (Uniswap | How Uniswap works, 2021). The number of each token they receive upon redemption may differ from the initial deposit as a result of *impermanent loss*, which will be explained at the end of this chapter.

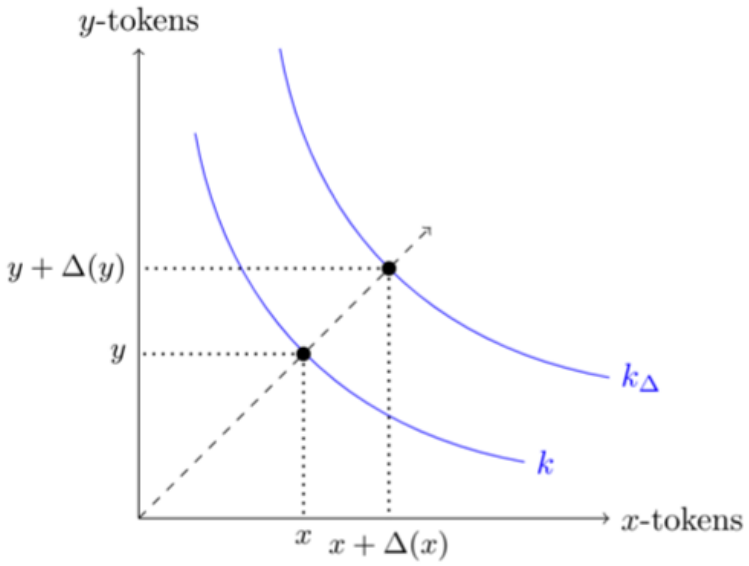


*Figure 2.13: Shifting 'k' with the CPM (Schär, 2020)*

Exchanges implementing liquidity pools as described above provide numerous trading opportunities and rely on traders to keep markets efficient. Different pools on different exchanges may have different exchange rates for the same token pair and different pools within the same exchange may have more favourable exchange rates for certain tokens when compared to their fiat prices. This opens numerous opportunities for trading which will be examined in this work.

Uniswap also allows the usage of the capital in the pools for a "*flash swap*", which is a type of "*flash loan*", a new type of financial instrument within decentralised finance. These will be described later in this work and utilised for the implementation.

Impermanent Loss (Impermanent Loss Explained | Binance Academy, 2021)

Impermanent loss occurs to liquidity providers when the market value of deposited liquidity changes compared to when it was deposited. For example, take a liquidity pool that currently has a 9 Token A and 900 Token B in its reserves, where 1 Token A is currently $100 and 1 Token B is pegged at $1. A liquidity provider, Alice, then provides 1 Token A and 100 Token B, giving them a 10% share of the pool. The value of their provided tokens in fiat total of $200.

In a scenario where Token A dramatically increased in value in the wider market to 1A/400B, arbitrage traders would add Token B to the pool and withdraw Token A, until the pool which begun at 1A/100B came into equilibrium with the wider market at 1A/400B. However, as the constant product model requires $x * y = k$, the product $k$ value (10,000) will remain the same (ignoring the small fees). Assuming the market is arbitraged into equilibrium, the value of one Token A is now 400 Token B within the pool, and for $k$ to remain constant at 10,000 this would require there to be 5 Token A and 2000 Token B within the reserves of the liquidity pool.

If Alice now withdraws their share of the pool, they receive 0.5 Token A and 200 Token B, where Token A is $400 and Token B is $1, totalling $400. This result could be considered a good profit from the starting value of $200, however in this case if Alice simply held the initial tokens, 1 Token A and 100 Token B the combined value would be $500. However, the effect of impermanent loss is often offset by the profits earned from the fees charged on trades which is included in the pay-out to liquidity providers upon redemption of their tokens.

This risk of impermanent loss has not deterred users, and at the time of writing, the popular decentralised exchange Uniswap has a total liquidity of over $7 billion (Uniswap Info, 2021).

### Smart Contract Reserve Aggregation

Smart Contract Reserve Aggregation works by aggregating liquidity reserves through a smart contract that allows large liquidity providers to connect and advertise prices for specific pairs, one example is the Kyber Network (Schär, 2020) (Luu and Velner, 2017). Prices are set by the liquidity providers, unlike liquidity pooling systems where they are determined in the contract (Schär, 2020). An entity who wants to exchange token $x$ to token $y$ requests a swap by calling the smart contract, which will compare prices from all liquidity providers and match the entity with the best price for their swap (Schär, 2020). This system functions well when there is a large number of liquidity providers however has drawbacks when there are few, this is due to collusion risks or monopolistic price setting (Schär, 2020). Often protection mechanisms are put in place to ensure prices are within a reasonable range. This protects users against negative consequences of mispriced trades. The details of this will not be covered in this work.

### Peer-to-Peer (P2P) Protocols

Peer-to-peer protocols allow users to negotiate the exchange rate of a swap bilaterally, with AirSwap (Oved & Mosites, 2017) being a widely used implementation (Schär, 2020). Participants first query the network for counterparties who would like to trade a token, followed by negotiating a price (Schär, 2020). This is automated to ensure efficiency. Often additional features such as off-chain indexers for peer discovery are implemented where these parties act as a directory in which people can advertise their intent to trade a pair (Schär, 2020). P2P protocols have the advantage of guaranteeing the price displayed for a swap is the price paid, unlike with liquidity pooling systems that suffer from slippage and non-linear returns.

Trading across all of these exchange types is not feasible in the scope of this work however should be considered for future research. The author focused on the automated market makers implementing the constant product model for trading.

### 2.4.3 Lending and Borrowing Systems

Lending and borrowing of on-chain assets is possible with decentralised finance and it is facilitated through *protocols for loanable funds* (*PLFs*) (Werner et al., 2020). This differs from peer-to-peer lending where funds are directly lent between parties. These protocols allow users to pool deposited tokens together in a smart contract (Werner et al., 2021). Any party may then directly borrow against the tokens deposited in the smart contracts, the "reserves" (Werner et al., 2021). This allows for both a lending and borrowing system with interest rates, allowing parties wishing to deposit money to earn interest and parties wishing to borrow assets to pay interest. One example is the popular Aave protocol.

#### Collateralisation & Liquidations

Collateralisation is defined by Werner et al. (2021) as "the process in which something of value is provided as security to cover the value of the debt." These protocols generally require over-collateralization for loans, as prices of tokens may fluctuate, to protect against under-collateralization and reduce risk (Werner et al., 2021). The problem arises that some tokens are volatile, and a significant price drop may cause the loan to become under collateralised (Werner et al., 2021). Due to this, a liquidation threshold is introduced. If a loan's deposited tokens value falls below the liquidation threshold a *"liquidator"* can liquidate the loan by purchasing the locked collateral at a discount, and often earns a bonus fee, to close the borrower's debt position

(Werner et al., 2021). This can be considered a form of arbitrage, as it introduces a situation where one can earn a risk-free profit by liquidating the loan, assuming there is no risk of failed transactions. In the case of the liquidator not having enough capital to close the under collateralised position, they may take advantage of new financial instruments introduced by decentralised finance such as flash loans (Werner et al., 2021). These will be explained later in the paper.

**Use Cases of PLFs**

The reader may be wondering why loans are used at all if they are over collateralised, however there are many incentives to both borrow and supply tokens to these protocols. On the supply side, the depositor earns interest which accrues per block (Perez et al., 2020) and can be withdrawn at any time of their choosing. On the borrowing side, it allows access to leveraged positions (Perez et al., 2020). One could deposit Ether as collateral and borrow DAI (a stable coin). One could then use this DAI to buy more Ether thus creating a leveraged position. If Ether appreciates, they benefit from the gain. The amount of DAI that would be required to pay back the loan would also fall relative to the value of Ether. A short position could also be created by borrowing an asset that one believes will depreciate in value and immediately selling it before repurchasing it at a later date to repay the loan (Perez et al., 2020). Tokens may also be borrowed for utility reasons such as governance tokens allowing the voting on future decisions of a protocol (Perez et al., 2020).

**Profitability of Liquidations**

Liquidating under collateralised loans tends to be particularly profitable as a trading strategy during market downturns. This is as a result of user's preference to use *stablecoins* as collateral for the loans and to borrow more speculative tokens. *Stablecoins* are often backed by a relatively stable asset such as the US Dollar, meaning their value does not wildly fluctuate. When the more speculative borrowed tokens appreciate, their value relative to the collateral rises meaning there are few opportunities for liquidating loans. In the opposite scenario, in the case of a downturn, the value of the borrowed tokens falls relative to the collateral leading to many loans becoming unhealthy, often suddenly. Cryptocurrencies historically appreciate for a period before being followed by sharp crashes. At the times of these crashes, large volumes of liquidations occur. This data is visualised in Figure 2.14 below and it can be seen as the Ether price falls sharply, which would be correlated with the general market sentiment, huge volumes of loans are liquidated.

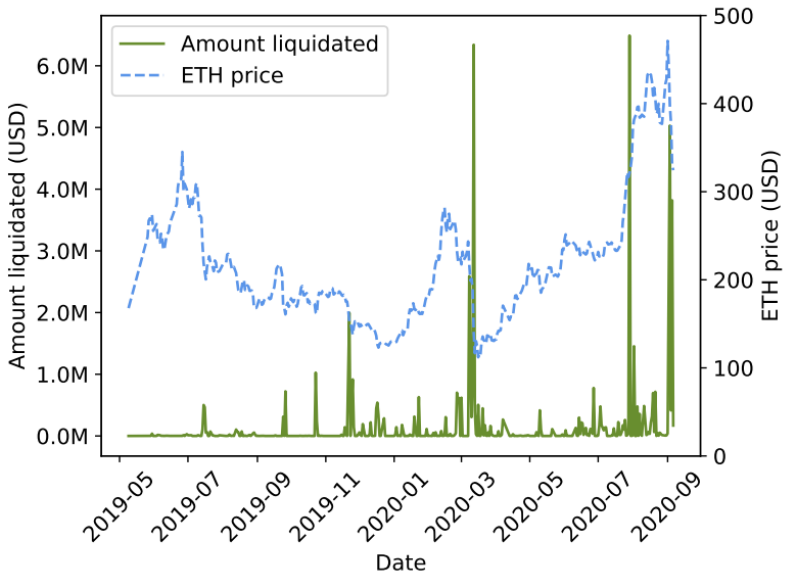


*Figure 2.14: Historic Liquidations and Ether Price Visualised (Perez et al., 2020)*

## 2.5 Novel Decentralised Financial Instruments

### 2.5.1 Flash loans

A flash loan allows a trader to receive a non-collateralised loan, requiring the loan to be repaid within a single transaction (Wang et al., 2020). There is no traditional finance equivalent, however, they function similarly to atomic transactions in traditional databases as the transaction is reverted if it cannot be completed with the loan being paid back within the transaction. Flash loans become useful for arbitrage opportunities, where a trader may detect a price difference among tokens on two different exchanges, acquire capital through a flash loan, complete a token swap between exchanges, pay back the loan and keep any remaining capital as profit. They are considered risk-free as if the loan is not paid back within the single transaction, the entire transaction will revert, meaning the trader will have lost no capital in the case of a loss occurring, bar the capital needed to pay gas (transaction) fees. There is also no risk of the loan not being repaid to the provider due to this mechanism. Flash loans can lend a considerable amount of capital (Wang et al., 2020) so even small price differences between tokens for arbitrage can yield large returns. Flash loans have also been a source of controversy and their risks scrutinised. Wang et al. (2020) investigated two incidents in 2020 occurring to *bZx* exchange that caused large capital losses through manipulating market prices resulting in attackers profiting to the tune of $829,500 and $1.1 million (Wang et al., 2020).

In the author's opinion, flash loans are simply a vehicle to provide capital and not inherently a problem and will lead to a more resilient ecosystem over time as actors using them maliciously will force developers to build more secure systems. Flash loans somewhat level the playing field within trading and allow anyone to access opportunities usually reserved for only those with large sums of capital.

## 2.6 Arbitrage

### 2.6.1 Overview

Stephen Ross (1989), the father of arbitrage pricing theory, defines an arbitrage opportunity as an "investment strategy that guarantees a positive pay-off." There are numerous forms of arbitrage in traditional asset and currency markets. This paper will focus on testing some existing currency arbitrage methods and investigating their performance in decentralised finance, along with exploring new types of trading and arbitrage which are possible due to the architecture and developments within decentralised finance. As decentralised cryptocurrency markets are unregulated and information asymmetries between different exchanges exist, many trading and arbitrage opportunities present themselves.

### 2.6.2 Historical Arbitrage Opportunities

Arbitrage is as old as currency itself. As mentioned by Yuval Noah Harari (2015) in his bestselling book Sapiens, in 1519 the conquistadors from Spain invaded Mexico. The natives, the Aztecs, quickly noticed the Spanish aliens who arrived took an interest in a yellow metal they used, gold. The Aztecs generally used cocoa beans or bundles of cloth for trading and did not value gold as highly. The natives questioned the Spaniard's passion for Gold and could not understand their willingness to trade valuable cocoa beans and cloth for it. The Spanish aliens were likely getting a much better deal buying gold from the Aztecs and bringing it back to Spain to sell than they would be swapping anything they had for gold in Spain. The Spanish arbitrageurs were buying goods at a low price and reselling them for a higher price in a different

market. This is an example of two markets being arbitraged into equilibrium and is one of the defining reasons currencies exist at all.

In the case of the Aztecs the markets likely did not come into equilibrium as the Spanish ultimately overthrew their empire, however, it is an example of early arbitrage, nonetheless. This exact mechanism driven by supply and demand happens today in markets in both traditional finance and decentralised finance markets and the act of arbitrage keeps them in a healthy equilibrium by eliminating information asymmetries.

### 2.6.3 Ethical Implications & Wider Implications of Arbitrage

It is widely accepted that for one party to make a profit in a traditional currency transaction, the other party must make a loss. This applies to arbitrage where another party may make an implied loss having not taken advantage of the arbitrage opportunity themselves. Although this may be seen as a negative impact, it is actually what internally regulates the markets into efficiency. The efficient market hypothesis (Fama, 1970), a cornerstone of financial academic literature, states a market is efficient when prices reflect all information available to participants. Therefore, for arbitrage opportunities to exist, there must be information asymmetry (Dothan, 2008). This is often between two different exchanges with different prices. The act of arbitrage trading brings markets back to a healthy equilibrium in most cases. Other arbitrage techniques such as front running have negative consequences on decentralised finance and will be discussed later in this work.

### 2.6.4 Traditional Finance Currency Arbitrage

In traditional finance, currency arbitrage keeps markets in a near-constant state of equilibrium, and arbitrage opportunities are acted upon at extremely high speeds with low latency to earn a risk-free profit. A true arbitrage opportunity is “risk-free”, so the trading methods discussed in this work are not arbitrage opportunities as there is the risk of losing funds due to failed transactions which incurs a fee. This is expected to change as the network matures and will be considered later in this work. Some traditional financial arbitrage methods are explored below.

**Buy-Low / Sell-High**

The idea behind this arbitrage method is to buy a currency on one exchange and then sell it for a higher price on another exchange, or for a different currency at a higher value. Generally, it is preferred to finish with the same currency the arbitrageur started with as if they were to convert at a later point in time back to the base currency, they would be exposed to exchange rate movements.

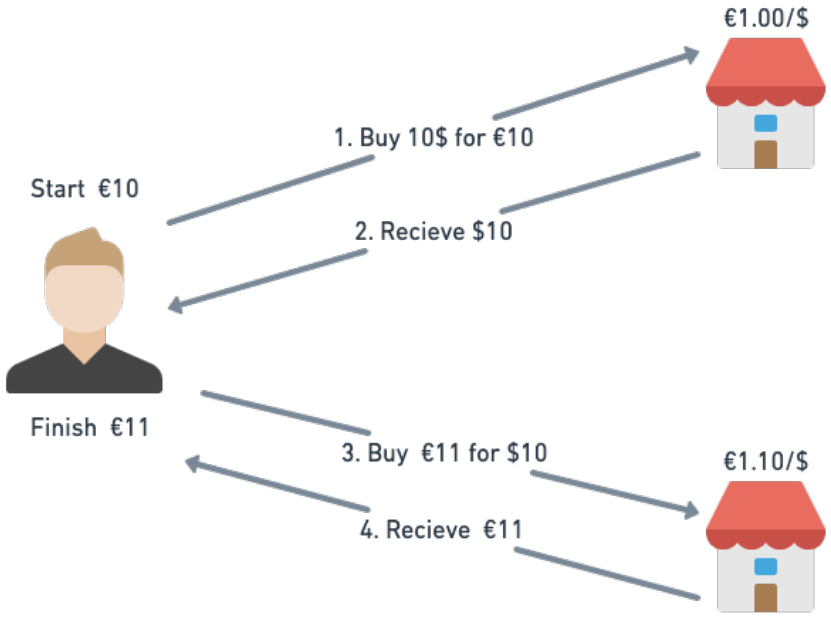


*Figure 2.15: Buy-Low Sell-High Arbitrage*

It is worth noting that this method applies to any asset that can be easily sold in two markets, not exclusively currencies. This includes goods and commodities arbitrage across countries to cryptoassets digitally. This method will be investigated in a decentralised finance context.

### Triangular Arbitrage

Triangular arbitrage uses at least 3 different currency pairs. For example, in traditional finance, a trader could exchange US dollars to some number of Japanese yen, then exchange the yen to some number of euro, and finally the euro back to US dollar (Aiba et al., 2002). If the arbitrageur is left with more US dollars than they had when they began the transactions, it is a profitable triangular arbitrage opportunity. To implement this algorithmically, we can represent the currencies, or cryptoassets, as nodes in a weighted directional graph and the edges between them annotated with the exchange rate. A diagram of such can be seen in Figure 2.16. An altered Bellman-Ford algorithm can then be used to detect negative cycles between currency pairs, finding profitable triangular arbitrage opportunities (Mahajan, Jadhav and Salian, 2020). A negative cycle is defined as a cyclic portion of a graph where the overall sum of the weights is negative, and the Bellman-Ford algorithm can easily be altered to detect these cycles.

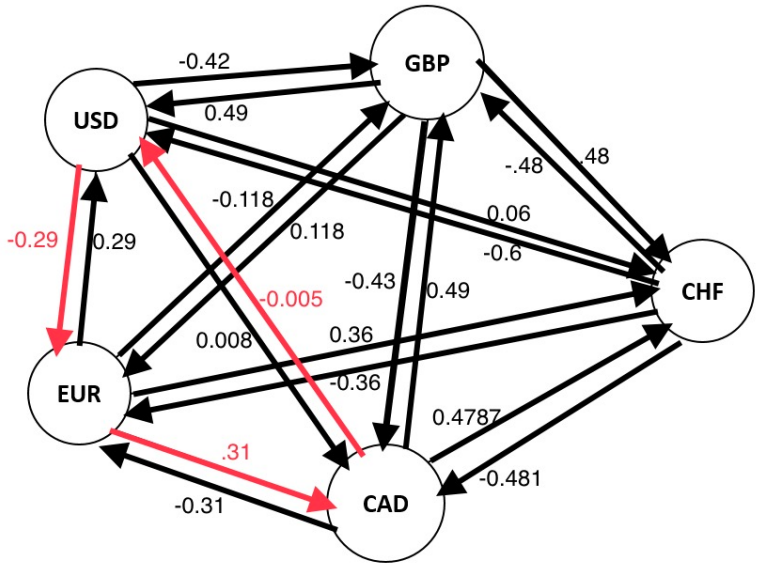


*Figure 2.16: Bellman-Ford Currency Arbitrage (TheAlgorists, 2021)*

A few other alterations are made, such as using negative logarithmic weights for the exchange rates. This allows detection of the negative cycles using the altered Bellman-Ford and as exchange rates are multiplicative the logarithmic weights allow for multiplication by summing the weights upon a traversal and the actual weight can be then recovered upon completion by exponentiating the sum (Martin, 2020).

Technically the arbitrager could finish in profit without making a triangle, similarly to the buy-low sell-high method, however the currency they finish on would then have to be priced based on the starting currency or converted at a later point in time in which case they are exposed to exchange rate movements.

### 2.6.5 Front Running

Front running in a traditional financial sense can be defined as trading an asset by a broker or person who has inside knowledge of a future transaction that is about to affect the assets price substantially (Investopedia, 2021). Front running is illegal in traditional finance and considered unethical. This is due to it being a zero-sum game. The success of the front runner and their profit from front running a transaction is directly proportional to the loss of the party they are front running (Anonymous, 2021). Due to the open network architecture of Ethereum and its Mempool which functions as a waiting area for transactions before they are processed, as visualised in Figure 2.7, front running transactions is possible and there are no legal penalties. A front runner may watch the Mempool for potentially profitable transactions, replicating a profitable transaction with a slightly higher gas fee to be quicker than the initial submitter of the transaction. This can be seen below in Figure 2.17.

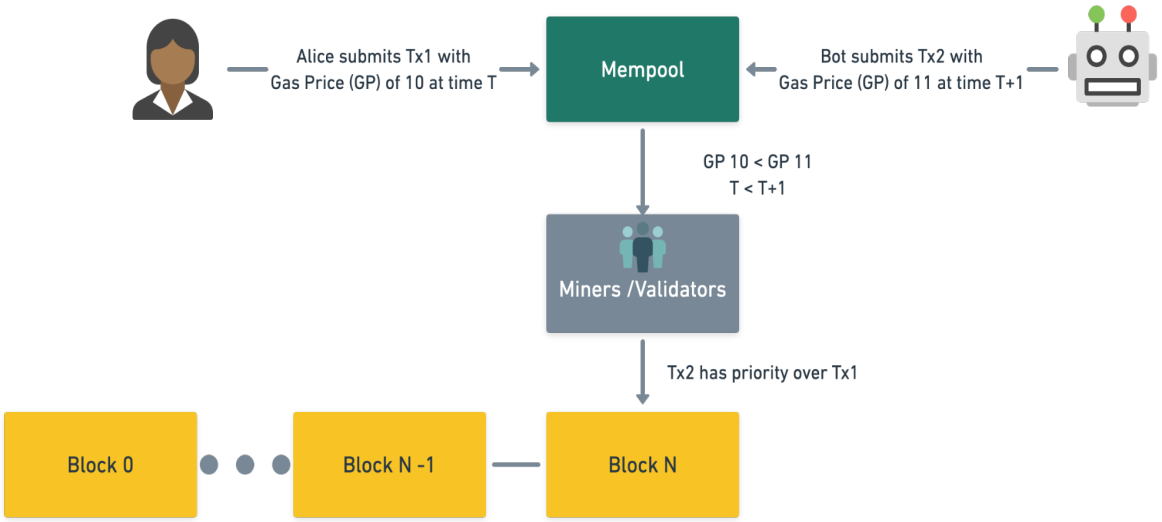


Figure 2.17: Generic Example of Front Running on Ethereum (Kisagun, 2019)

Bots engaging in front running may also watch for large transactions which will shift the price of an asset substantially and place a strategic trade to profit due to the large movement in price. This will be at the expense of the party submitting the initial large transaction. The Mempool is watched by numerous bots which engage in front running, and the miners who mine the transactions also make additional profit by front running transactions or other mechanisms which will be discussed later in this work. This is as a result of the network allowing arbitrary ordering of transactions within a block at the discretion of the miner who mines the block.

**Gas Implications (PGAs)**

The bots who engage in front running will also be competing against each other. This leads to a dynamic bidding up of transaction fees to obtain priority in the transaction ordering (Daian et al., 2019) to profit from a potential transaction. This was defined by Daian et al. (2019) as a priority gas auction (PGA). This increases transaction fees across the entire network due to congestion and ultimately is a negative consequence of front running on other parties submitting transactions on Ethereum. Even if the party is not front run themselves, they will pay a larger transaction fee due to the gas fees being bid up through priority gas auctions. This is a major problem with the implemented trading methods as high gas fees will have to be taken into account which will lower profit margins. There is also the risk of being front run or being too slow to act upon the opportunity and the transaction being rejected, which still consumes gas and incurs a fee.

**Game Theory & Collusion**

Daian et al, (2019) found bots colluding to maximise their profits through 'grim trigger' cooperative strategies. In the previously explained priority gas auction system, each bot will submit a transaction with a marginally higher fee than the transaction they are attempting to be faster than to gain priority on their transaction being mined first. If a bot is to 'misbehave' and try to significantly increase the gas price to be quicker, another bot in the auction will raise its gas price to an amount where any higher would be unprofitable thus eliminating the profitable opportunity for all bots. The expected payoff from being part of the cooperative strategy and only raising the minimal amount is larger than the payoff from deviating from it. Daian et al. (2019) show under these conditions there exists a Nash equilibrium for cooperative strategies, essentially leading to large numbers of bots colluding to optimise profitability.

**Miner-Extractable-Value (MEV)**

The additional value that is extracted by miners who validate the transactions, excluding transaction fees, is defined by Daian et al. (2019) as *miner-extractable value* (*MEV*). There are numerous mechanisms for miners to extract additional value, such as offering ordering optimisation (OO) for a fee or by engaging in malicious practices themselves. Ordering optimisation allows submitting of a transaction or a bundle of transactions within specific positions of in block. This is displayed in the example of front running in Figures 2.18 – 2.21 where it is clear the front runner has had some influence on the order of transactions in the block to ensure they could profitably front run the transaction. This area of front running, collusion and attacks is commonly referred to as the "Dark Forest", which is a reference to the popular science fiction novel "The Dark Forest" authored by Liu (2008).

Robinson (2020) describes the concept of a "dark forest" from the book as "an environment in which detection means certain death at the hands of advanced predators" and in the context of the Mempool, the malicious bots are the predators which monitor pending transactions and attempt to exploit profitable opportunities created by them. Qin et al. (2021) investigated how problematic this is by building a generalised trading bot using an automated transaction replay algorithm capable of replicating unconfirmed profitable arbitrage transactions, front running them and back testing it on previous blockchain data. Qin et al. (2021) estimated this bot would profit by over $17 million over the 2 previous years examined.

**Private Pools**

Private pools are another systemic risk. Qin et al. (2021) estimate that 1.64% of miner's transactions are not broadcast to the network and instead mined privately. This offers the unique

opportunity to protect against front running since potential front runners do not see the transactions before they are confirmed on the blockchain. Qin et al. (2021) show that there is evidence that the largest Ethereum mining pool performs front running itself and cloaks its private transaction mining activities. As displayed in Figure 2.21 and discussed later in this work, there is also evidence that this is sold as-a-service to front runners who pay for transaction ordering optimisation and privately mined transactions. As Ethereum is often thought of as an open decentralised network, this is a stark contradiction to that assumption.

**Flashbots**

Flashbots is a research and development organisation (Obadia, 2020) with the goal of mitigating the negative externalities and existential risks posed by miner-extractable value (MEV). They have proposed a permissionless, transparent and "fair" ecosystem for MEV extraction to preserve Ethereum's core values of openness and decentralisation (Obadia, 2020). At the time of writing, they have designed and implemented a proof of concept for permissionless MEV extraction called MEV-Geth that is a sealed-bid block space auction mechanism for communicating transaction order preference (Obadia, 2020). This allows one to submit a transaction or a bundle of transactions that bypasses the Mempool and only incurs a fee if selected to be included in that block, which eliminates failed transaction risks. It also allows the miner who mines the transaction to earn additional revenue through the fee for the service. Although this may be thought of as an implementation of the exact problems the author listed above, Flashbots take the view that MEV extraction and these practices are inevitable, so they have the objective of bringing the issues to light, democratising MEV extraction and distributing benefits to align the interests of miners, traders and developers (Obadia, 2020).

**Front Running Example**

Using Etherscan, a tool to examine the Ethereum blockchain which includes blocks and transaction data, the author painstakingly sifted through recent blocks on the chain for examples of front running. One example can be seen in Figures 2.18 – 2.21, the details of which will be explained along with the likely method the front runner used to achieve this. The exchange used for the trade is Uniswap, which falls under the automated market maker category (AMM) and uses a constant product model for pricing tokens. As considered previously, the exchange rate changes after every transaction that is governed by the constant product model and is affected by the liquidity in the pool. The front runners' address is 0x7d92ad7e1b6ae22c6a43283af3856028cd3d856. The front runners' '*sandwich attack'* transactions are seen below in Figure 2.18, Figure 2.20 and Figure 2.21, and the unfortunate user who was front run is seen in Figure 2.19.

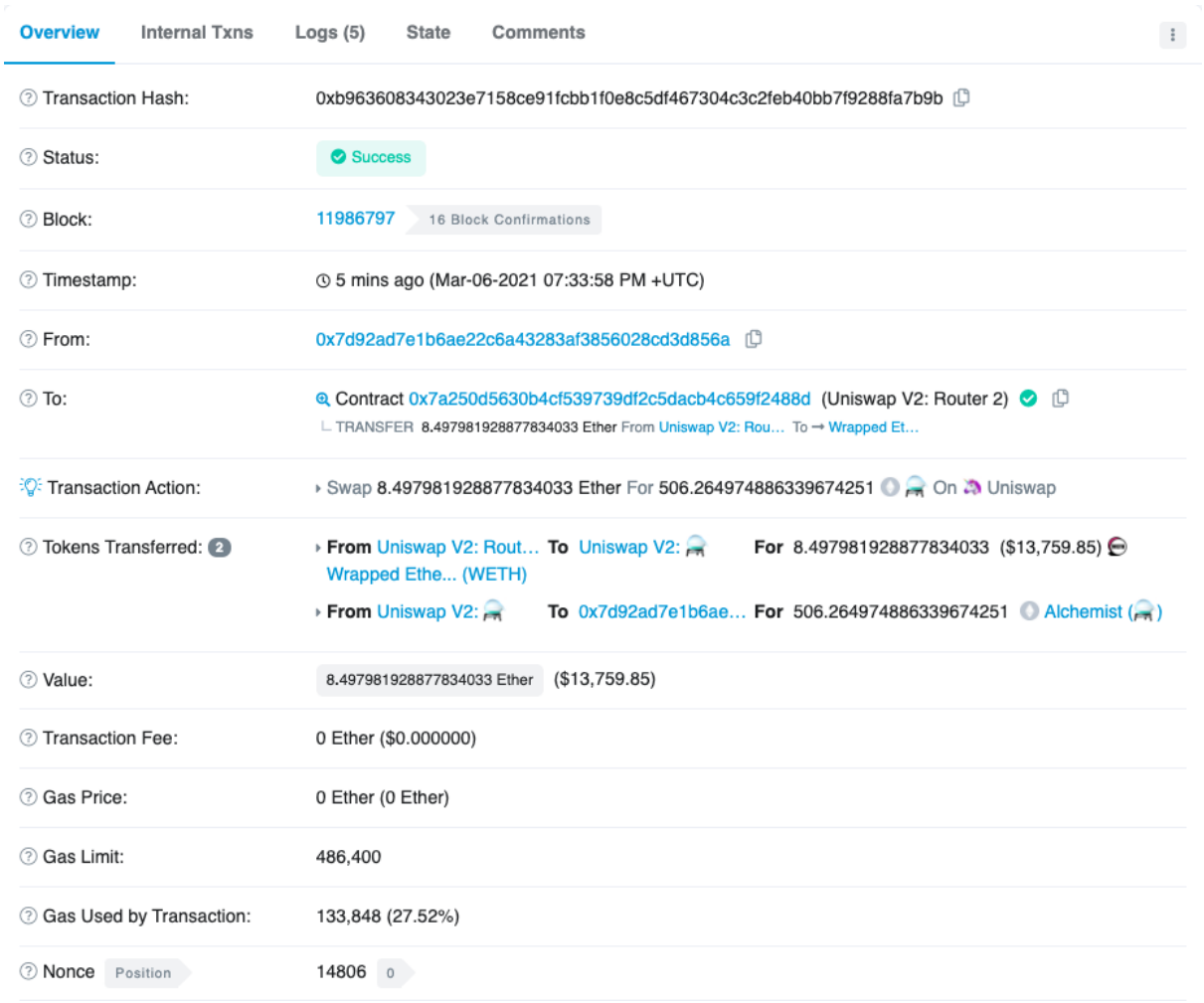


*Figure 2.18: Front Running Example - Transaction 1*

In the first transaction, the front runner swapped $13,769.85 in WETH to Alchemist Token. It is worth noting that this is the first transaction in the block, perfectly placed before the non-front runners' transaction which occurs next. This removes Alchemist Token liquidity from the pool, altering the exchange rate. It is now more expensive to obtain Alchemist Token from this pool due to the decreased liquidity. The front runner has now also obtained some Alchemist to swap back to WETH after the non-front runner significantly alters the exchange rate with their transaction so the front runner can profit off of the movement.

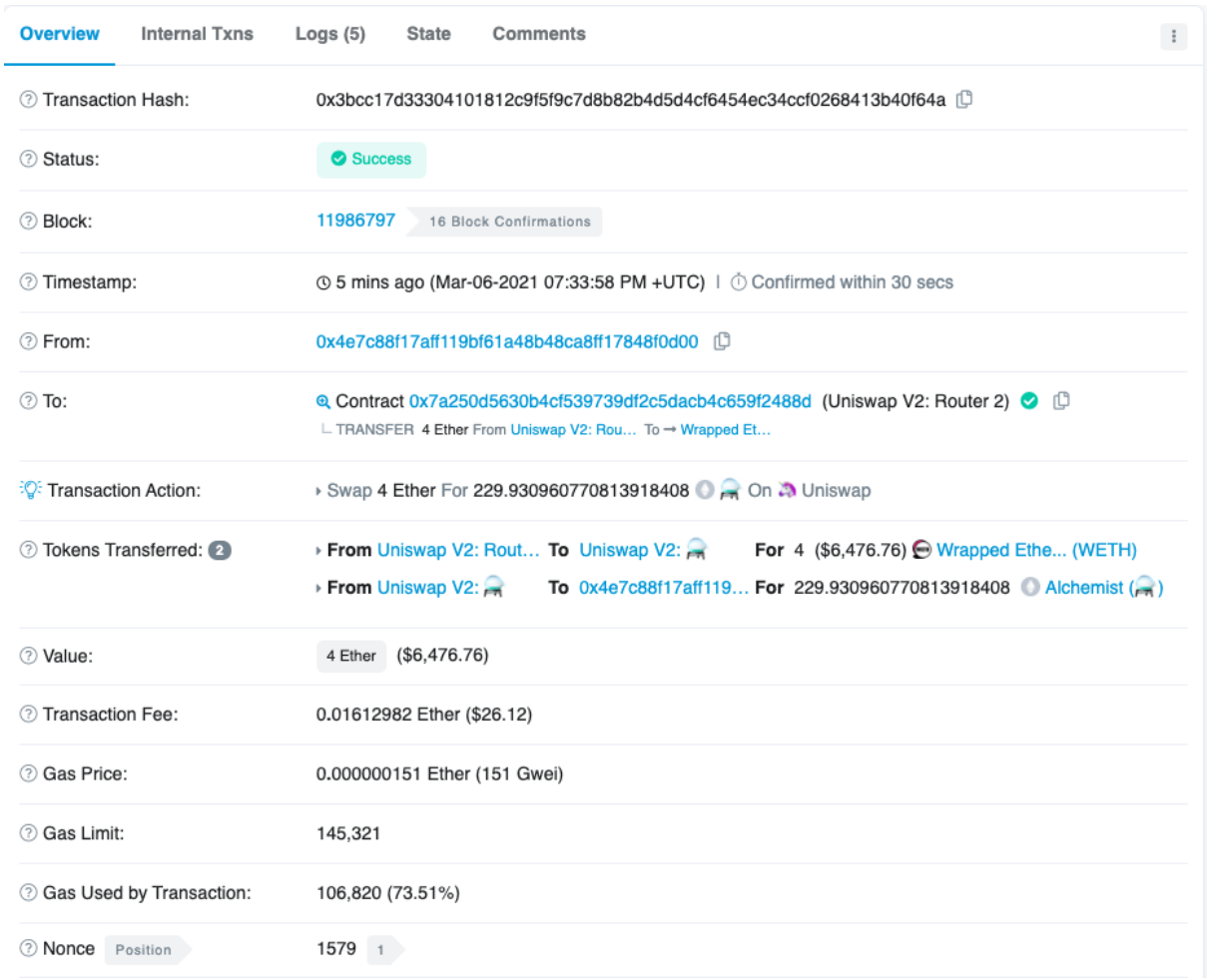


*Figure 2.19: Front Running Example - Transaction 2*

In the second transaction, the non-front runner has submitted a transaction to swap $6476.76 in WETH to receive as much Alchemist token as they can. As the pool price was significantly shifted in the previous transaction as Alchemist is not a popular token and has low liquidity, they now pay a significantly higher price than initially expected, along with further altering the exchange rate which the front runner will take advantage of. The loss incurred by the non-front runner by making this trade is directly proportional to the front runner's profit.

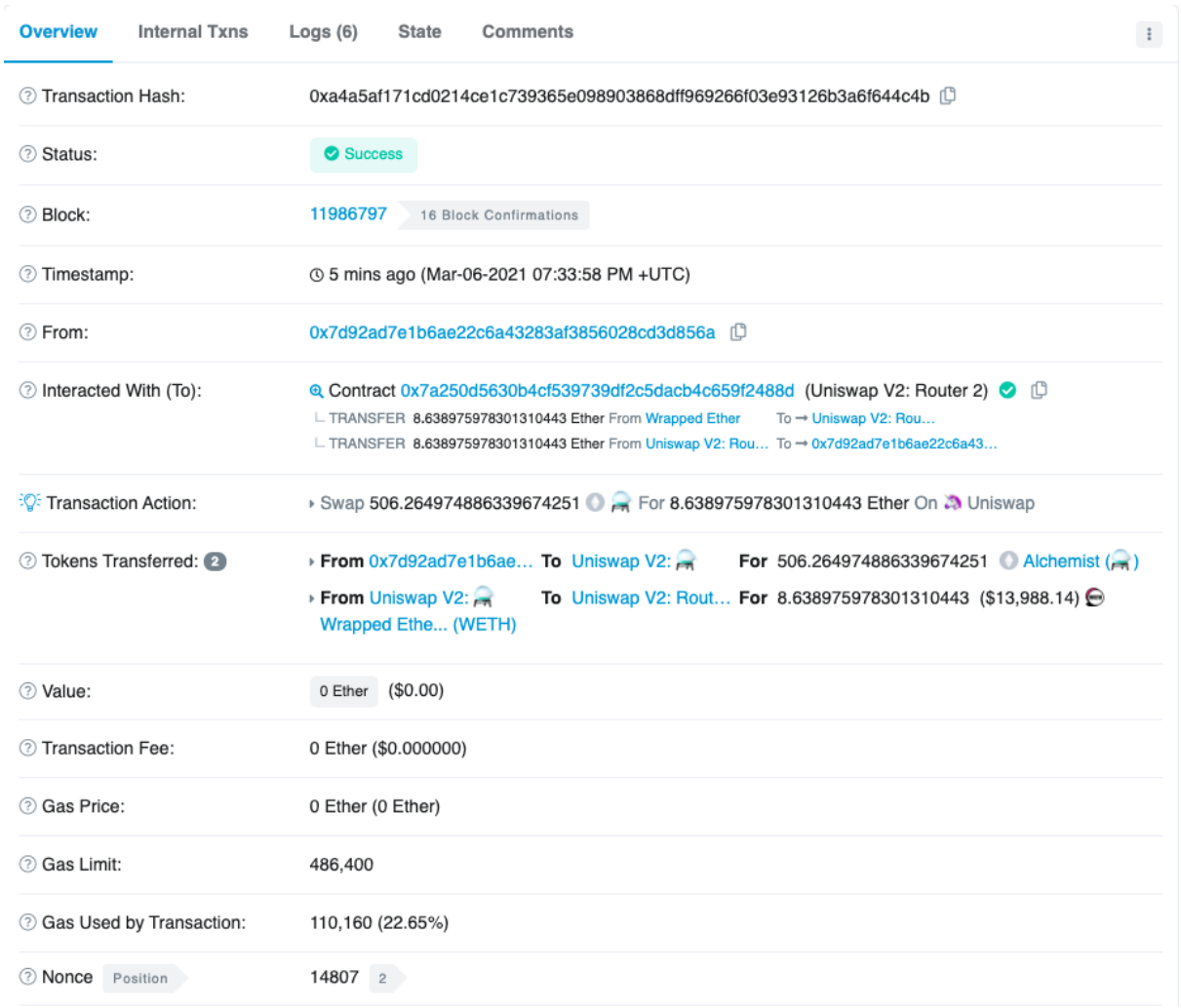


*Figure 2.20: Front Running Example - Transaction 3*

In the third transaction, the front runner swaps their Alchemist token back to the starting token at a more favourable exchange rate. He receives $13988.14 in WETH. This is a significant profit of $228.29 from their starting amount. It is worth noting here that the front runner's transaction fees have all been $0 along with a gas price of 0 Ether. This is not normal for Ethereum and suggests access to some service where miners allow both priority ordering and payment in some form that is outside the standard gas fee mechanism.

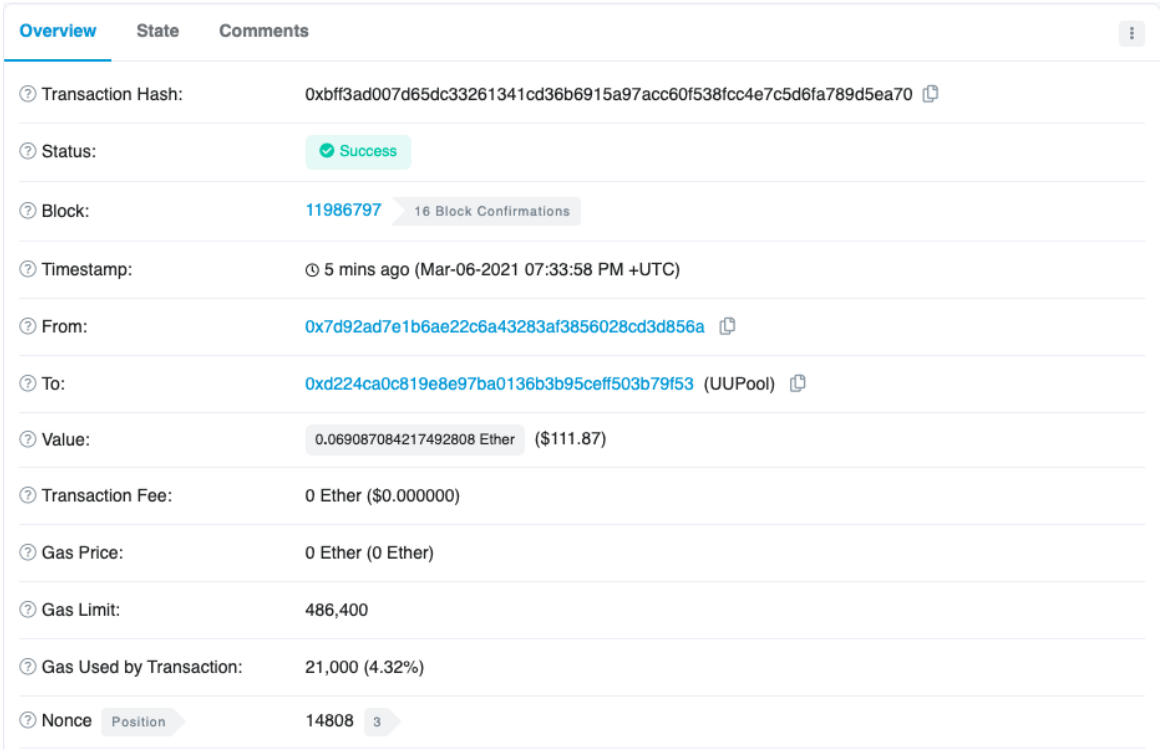


*Figure 2.21: Front Running Example - Transaction 4*

In transaction 4 we can see the front runner sends some Ether, again at no fee, to an address called UUPool. The transaction fee (gas) of the non-front runner in transaction 2 was ~$26 which would be considered standard in the Ethereum network at the time of the transactions, while the gas fees of the front runner's transactions are all valued at near $0. The transactions are also perfectly placed in the block for such a front run to occur, with the transactions being in the 0th, 1st, 2nd and 3rd position. This would lead us to believe that the front runner has some control over the placement of the transactions in the block or pays for it as a service. There is a possibility that transaction 4 is a payment for such service, as here $111.87 is paid to an account and upon an investigation of this account, numerous payments were sent following other front running transaction sequences. Including the pay out in transaction 4, Figure 2.21, as a cost, the net profit for the attacker was $116.42. This type of attack is commonly referred to as a *sandwich-attack* (Zhou, 2020).

The author expects front running and MEV extraction techniques like ordering optimization to have a significantly negative impact on the success of implemented trading methods. As mentioned by Foxley (2021), one of the largest Ethereum mining pools Ethermine have even started to promote "*front running as-a-service*" which is problematic as they control over 20% of the mining power at the time of writing (Best Ethereum Mining Pools for 2021 - PoolWatch.io, 2021). The three largest mining pools collectively control over 50% of the mining power (Best Ethereum Mining Pools for 2021 - PoolWatch.io, 2021) and it is assumed that all of these pools attempt to maximise their profits by engaging in front running or selling the functionality as a service. One could also argue the security of the network is threatened here and it is not truly decentralised. This large share of mining power combined with the knowledge of front running services implies that if our trading strategy identifies a profitable opportunity, and is faster than every other trader, and overcomes the other obstacles which will be discussed later in this work, if it is detected by these services who mine 1 in 2 transactions, they will likely take the opportunity themselves.

However, what if they do not see the trade or there are ways to avoid detection? The author will explore potential solutions and attempt to find profitable opportunities later in this work by looking where other traders and miners are not.

**Combatting 'Sandwich Attacks' using Malicious Code**

Combating front runners as a user who is interacting with decentralised applications through their web interfaces can be difficult. The architecture of the network due to the Mempool and mining allows front running to thrive. In particular, the *sandwich attacks* mentioned have become extremely widespread, with large mining pools such as Ethermine now offering '*front-running as-a-service'* to extract additional value for miners (Foxley, 2021). During the author's research, he uncovered a clever piece of code to earn a risk-free profit, using malicious code that no normal user would interact with, only the front runners. This is worth mentioning as it offers a glimpse into a potential way to earn a risk-free profit at the expense of the front runners. Front running on Ethereum is neither illegal nor against any rules, however, the author considers malicious front running unethical as it is essentially gaming the architecture of the blockchain system at the expense of its users. Making a profit at the expense of front runners, who are manipulating the wider ecosystem, seems justified.

Worsley (2021) noticed the generalised nature of most front running setups, essentially repeating the attacks shown in Figure's 2.18 – 2.21 by targeting Uniswap trades with significant trade sizes. In particular, Ethermine appeared to be the main culprit, and their sandwich trading account had a balance of over 100 Ether. Worsley (2021) decided to deploy a standard ERC20 token but with one modification, it had extra logic to detect when anyone other than the deployer is transacting with it. In these situations, it only returns 10% of the specified amount, despite emitting event logs which makes it appear it traded the full amount. The poisonous transfer function can be seen below in Figure 2.22.

```
function _transfer(address sender, address recipient, uint256 amount) internal
virtual {
  require(sender != address(0), "ERC20: transfer from the zero address");
  require(recipient != address(0), "ERC20: transfer to the zero address");
  uint256 senderBalance = _balances[sender];
  require(senderBalance >= amount, "ERC20: transfer amount exceeds balance");

     /* Malicious Code below */

  if (sender == ownerA || sender == ownerB) {
    _balances[sender] = senderBalance - amount;
    _balances[recipient] += amount;
  } else {
    _balances[sender] = senderBalance - amount;
    uint256 trapAmount = (amount * 10) / 100;
    _balances[recipient] += trapAmount;
  }

  emit Transfer(sender, recipient, amount);
}
```

*Figure 2.22: Poisonous Transfer Function implemented by Worsley (2021)*

Worsley (2021) then deployed this contract and set up a Uniswap pool containing both Ethereum and the new ERC20 token with the malicious transfer method. Following this, Worsley launched a series of bait transactions that were sufficiently large to attract attention of front runners, and with a low transaction fee so they would be slow to be confirmed by the network. This allowed the front runners sufficient time to execute their sandwich attack. As the sandwich attack involves transferring the malicious ERC20 token, the malicious transfer function would be executed. The front runners executing the sandwich attacks took the bait, and Worsley (2021) shows transactions on Etherscan showing the profit from this strategy, with two of his transactions alone profiting a combined sum of over 100 Ether ($170,000 at the time of writing).

### 2.6.6 Centralised Cryptocurrency Exchange Trading & Arbitrage

#### Inter-exchange Trading & Arbitrage

This type of trading takes advantage of the pricing spreads between exchanges. It was extremely popular and an efficient method of arbitrage during the period of interest in cryptocurrencies during 2017. As described by Han (2018), one method of inter-exchange arbitrage is simply by building a bot to monitor price differences between exchanges and when a discrepancy is found, buying the currency at a low price on one exchange and selling it at a higher price on another exchange. This is described previously as a "*buy-low sell-high*" trading strategy. Trading and transfer fees are also taken into account and the trade is only executed if it will be profitable. This requires a considerable amount of the starting capital to effectively take advantage of the price discrepancies. A slightly altered version of this *buy-low sell-high* method will be implemented in this work to achieve this within the context of decentralised finance with minimal starting capital.

#### Intra-exchange Trading & Arbitrage

Intra-exchange trading and arbitrage involves making a sequence of trades within a single exchange and starting and ending on a specific currency but finishing with more of the currency than you had before initiating the trades (Han, 2018). As described above, this is common in traditional finance and can also be applied in a decentralised finance context.

## 2.7 Upcoming Ethereum Changes

Upcoming changes to Ethereum that will affect any future implementations of the novel trading and arbitrage methods implemented in this work will be considered in this section.

**Rollups**

As will be explained later in this work, a massive increase in usage has caused a plethora of scaling problems for Ethereum, such as higher transaction fees which adds a financial risk to the implemented trading methods. If this fee did not exist, they would be pure arbitrage opportunities. The solution to this is to change the blockchain itself to allow for a higher transaction capacity by using a more efficient consensus mechanism, such as Proof-of-Stake, or by using techniques such as *sharding*, which splits the blockchain into smaller sub-chains (Buterin, 2021). However, building, testing and implementing these upgrades takes time on such a large distributed system. In the meantime, rollups have offered a temporary solution. A rollup involves moving computation and some state storage off-chain but keeping some data per transaction on-chain for security (Buterin, 2021). They use fancy compression tricks such as replacing data with computation when possible. There are two main types of rollups, *ZK-Rollups* ("zero-knowledge-proof Rollups") and *Optimistic Rollups* (Buterin, 2021). These are quite complex and will not be described in detail, however as they are due to be implemented very soon at the time of writing (April 2021) the near future implications will be considered. Essentially, the efficiency increase should lower transaction fees, including failed transaction fees. This dramatically reduces the risk of the implemented trading methods and should lead to increased profitability, although more traders and arbitrageurs may appear after the changes making the space more competitive.

**EIP-1559**

Ethereum Improvement Proposal (EIP) 1559 is due to be implemented in July 2021 to combat the recent surge in transaction fees (gas prices). It changes the traditional method of paying the gas fee directly to the miner to a new mechanism in which the gas fee is sent to the network itself and *"burned"* (destroyed) with only an optional tip paid to miners (Foxley, 2021). This has large support from the Ethereum community but is opposed by miners who will lose revenue. Miners protested this and even planned a 51% attack to protest, which would comprise the network (Foxley, 2021). This could lead to some short-term security concerns however if this is successfully implemented in July 2021, it should lower transaction fees reducing risk for the implemented trading methods. However, on the flip side, large pools such as Ethermine have added front running software to help miners offset lost revenues from EIP-1559 (Foxley, 2021). This software could negatively impact the viability of the implemented trading methods and make it an insider's game, with miners replicating and replacing any profitable transactions as they choose.

**Eth 2**

Eth 2 is an upgrade to the Ethereum blockchain to enhance the speed, efficiency and scalability of the network (Millman, 2021). The first node of phase 1 was successfully launched on December 1st, 2020 (Daws, 2021). The current Ethereum network supports approximately 30 transactions (Millman, 2021) per second and is at full capacity, which has caused a large increase in transaction fees. Eth 2 promises up to 100,000 transactions per second upon full completion. This should significantly lower fees and reduce the amount of lost capital for failed transactions when missing trading opportunities or being front run. Eth 2 is currently operational however the full implementation date is not yet confirmed. It is expected to be fully implemented by 2022.

# Chapter 3 | System Design

## 3.1 Approach

To investigate novel trading and arbitrage methods within decentralised finance, the author implemented several trading strategies across different protocols within the decentralised finance landscape. Some took inspiration from traditional finance while others are unique to the decentralised financial system. The goal was to figure out the viability of the novel trading and arbitrage methods using the Proof-Of-Concept implementation and come to a conclusion about the potential profitability of the methods if they were to be implemented optimally. This involved researching numerous protocols, learning their architectures, researching the new financial instruments, learning their implementation, learning how to develop smart contracts and integrate with other protocols within the smart contracts on-chain, and build scripts to monitor opportunities, work out profitability and trade sizing and execute upon the opportunities where possible.

Due to the front running uncovered in research, the author was not expecting the methods to be profitable when implemented on popular protocols. However, the author wanted to evaluate how competitive the landscape is and find ways to beat the competition by applying the methods in different scenarios. Although the majority of the methods were developed on popular protocols (due to reliability and development documentation) adapting the strategies to target lesser-known protocols is possible and will also be implemented in this work.

## 3.2 Alternative Approaches & Methods

### Usage of Other Blockchains

At the time of starting research and development, decentralised finance was almost exclusive to the Ethereum ecosystem. Other blockchain teams quickly noticed the popularity and built new blockchains supporting smart contracts, such as the Binance Smart Chain (*BSC*). BSC is particularly interesting as it is almost a direct fork of Ethereum but implements a different consensus mechanism known as Proof-of-Stake-Authority (Peaster, 2021). This provides access to decentralised finance with significantly lower fees, at the expense of network security. Security is considered lower as the network is not fully decentralised, with a meagre 21 validators supporting the network (Peaster, 2021). As it is essentially a fork of Ethereum, it also supports the Solidity programming language. This led to many developers forking popular protocols such as Uniswap and releasing them on BSC under new names. Usage surged in early 2021.

The methods implemented by the author in this work could be easily adapted to target blockchains such as BSC in the future. Due to the significantly lower fees on these blockchains, the risk from failed transactions would also be reduced. There is also the possibility of less competition and front running. It is worth noting that the Ethereum upgrades described in Chapter 2.8 should reduce the fees to a similar level, however in the meantime trading may be more profitable on other chains such as BSC. The author did not have time to test the implementation on BSC as flash loans were only implemented on BSC in March 2021, however, the code could be adapted now to run on BSC with minimal effort.

### Trading and Arbitrage without Flash Loans

Flash loans are used throughout the implementations to provide access to a large amount of capital. These flash loans do have a cost which in turn reduces profitability. In the context of future work, it would be more profitable to not use flash loans if access to large amounts of capital was available. The risk could also be reduced of unprofitable transactions by simply reverting at the end of the transaction if the trades were not profitable, similarly, to how is done using a flash loan.

### Alternative Trading and Arbitrage Methods

There are numerous alternative trading and arbitrage methods available when access to additional capital is present. One example is arbitrage between centralised exchanges and decentralised exchanges. Upon noticing a price difference between exchanges, if one possessed both tokens on both a centralised exchange and in a wallet that could be used on a decentralised exchange, the tokens could be simultaneously swapped from one token to another and if there is a price imbalance that is significant enough to cover gas fees and price impact on the decentralised exchange this would lead to a profit.

More traditional finance methods could be considered also such as cash-and-carry trades where an investor simultaneously enters a long position in an asset while selling its associated derivative by shorting a futures or options contract (Ganti, 2021). Derivatives are still in their infancy in the decentralised finance space however the author predicts these methods will blossom as the technology advances in the coming months.

## 3.3 Methods Implemented

### Flash-Low Sell-High

The "*Flash-Low Sell-High*" strategy is an augmented version of the traditional *buy-low sell-high* trading strategy. The author implements and investigates this method in Chapter 4.3.

### Flash Liquidation

As described in Chapter 2.4.3, liquidating under collateralised loans can be a profitable trading strategy. The "*Flash Liquidation*" section implements this trading strategy in Chapter 4.4.

### Flash Buy-Low Sell-High

This method is similar to the *Flash-Low Sell-High* method but allows the usage of two different exchanges rather than being locked to Uniswap. It can be thought of as a traditional *buy-low sell-high* method using a flash loan as the initial provider of capital that is then reimbursed at the end of the transaction. The author describes his profitable implementation of this method while keeping the underlying strategy secret in Chapter 4.5.

### Triangular Trading

Triangular arbitrage was discussed in the context of traditional finance in Chapter 2.7.4 however it can also be adapted for a decentralised finance context. Development of this method was commenced but remains unfinished due to the complexities discovered which will be discussed and time constraints of this research. The author presents his unfinished implementation and evaluates the problems uncovered before describing a potential solution that would generalise across all exchange types in Chapter 4.6.

## 3.4 Requirements

### 3.4.1 Functional Requirements

#### Implementation of Novel Trading Methods

The author identified four novel trading methods to investigate, the "*Flash-Low Sell-High*" strategy, the "*Flash Buy-Low Sell-High*" strategy and the "*Flash Liquidate*" strategy. These are methods that are unique to the decentralised finance ecosystem. "*Triangular Trading"* was also investigated.

#### Operational Usage of New Financial Instruments Exclusive to Decentralised Finance

Flash loans, as described in Chapter 2.6.2 are ultimately what allowed the author to implement the novel trading methods by providing access to the needed capital. Flash loans will be implemented and tested for successful functionality.

#### Pricing Optimisation

From the authors research it was discovered that this can be a very competitive environment when it comes to successfully executing on the opportunities, so it was paramount the development of the scripts to price the trades functioned correctly so data could be gathered on potential profitability prior to deploying and executing the trades on the Ethereum Mainnet.

#### Scripts to Evaluate the Implementations

The author needed to be able to gather and evaluate data regarding the potential profitability of the strategies prior to testing them in a production environment on the Ethereum Mainnet. The author also wanted to be able to visualize the gathered data.

### 3.4.2 Non-Functional Requirements

**Reliability**

As the scripts would be deployed on a server if operating under optimal conditions (see Chapter 3.6, optimal architecture) they should be reliable in carrying out calculations and executing transactions without crashing. They should also not execute on unprofitable opportunities. There is still the risk of failed transactions however these error cases should also be handled to stop the program crashing.

**Performance**

As time is quantised in blocks when dealing with the blockchain, execution speed is not as influential as it is in other trading systems such as High-Frequency Trading (HFT), however a base level of speed is needed. The transactions should be executed efficiently as possible with the Proof-Of-Concept system. However, the Proof-Of-Concept system will still have performance problems due to the network architecture which are detailed in Chapter 3.6. When using an optimal architecture and executing this in a production environment (see Chapter 3.6, optimal architecture) the system should perform sufficiently.

**Security**

The contracts should be secure in the sense that only the deployer of the contract should be able to invoke any methods. This prevents potential front runners from easily copying the trade using the authors deployed contract, however it does not protect against generalised front running where a front runner may have a similar contract deployed that they can utilise.

**Flexibility**

As Solidity contracts are size constrained by the Ethereum, and expensive to deploy as they grow, a flexible contract that could call any exchange is impossible at the time of writing. However, the contracts should be flexible in the sense that the methods implemented could be easily adapted to target different protocols by slightly modifying the code.

## 3.5 System Architecture

The author implemented and tested the trading methods using the Proof-of-Concept architecture detailed below in Figure 3.1. However, the author was aware that this architecture has numerous limitations which will be considered. Two further system architectures are proposed, an improved architecture that would be reasonably easy to implement assuming access to extra storage space or capital to acquire some, and an optimal architecture if the trading methods were to be run on a system built in an optimal manner to maximise the potential profitability of the implemented trading methods.

**Proof-of-Concept Architecture**

The architecture that will be implemented for a proof-of-concept system will be using a locally hosted server running the trading bots to detect opportunities. When these are detected, the bot will submit a transaction via an API to a service such as Infura. Infura hosts numerous blockchain nodes to allow interaction to the blockchain over API.

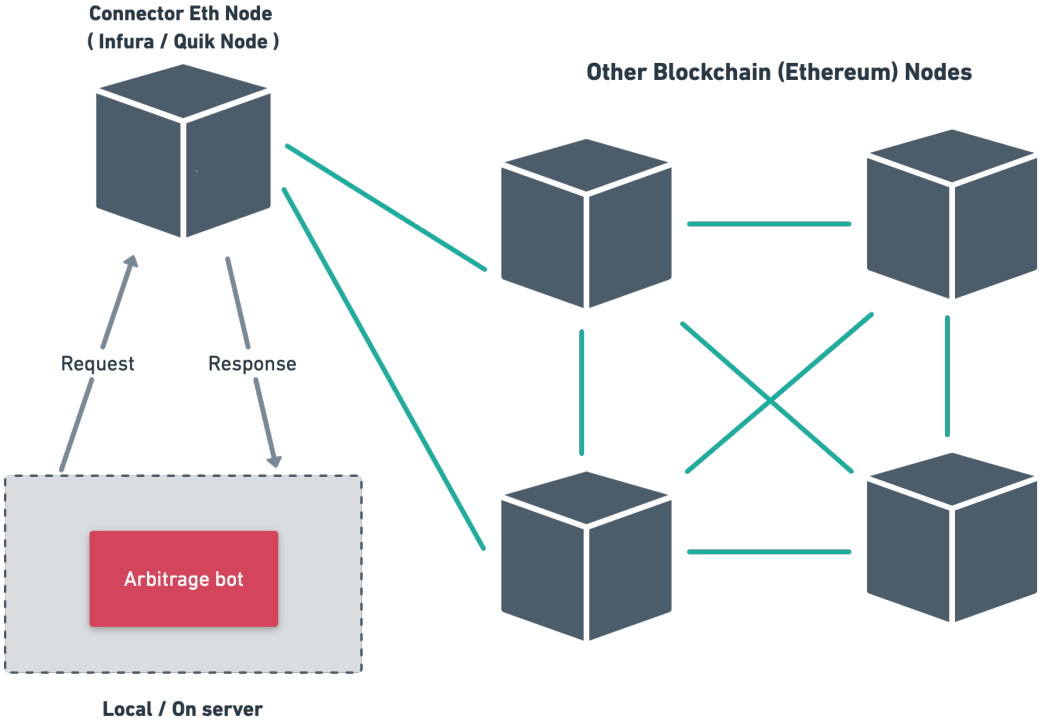


*Figure 3.1: Naive System Architecture (PoC)*

This system has some flaws as although there are near constant block times in the Ethereum network, one is added to the chain roughly every 15 seconds, the network is not perfectly in sync. Latency on the network can be a determining factor to which party submits the profitable transaction first. This is the result of the latest block not always being available to the blockchain node our script is connected to, so it may end up in a situation where we detect the opportunity later than other nodes, and subsequently act on it late. This is the result of the time taken for a new block added to the blockchain to propagate from the miner's node across the network to the node the script is interacting with. There is also the risk of being front run by other traders or the mining pools as described previously.

## Improved Architecture

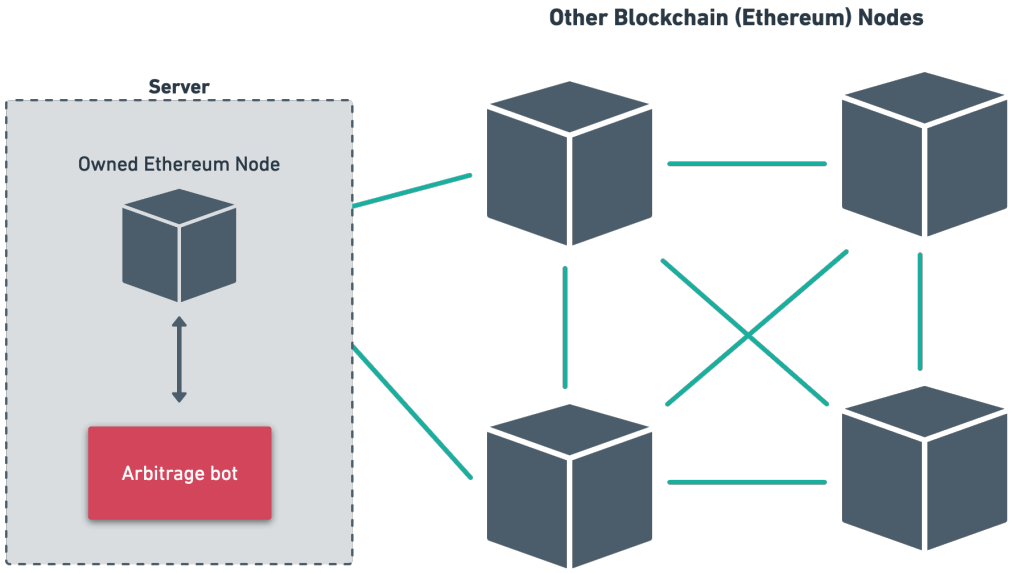


*Figure 3.2: Improved System Architecture*

By placing the arbitrage bot on the same server as an Ethereum node, as seen in Figure 3.2, some latency is reduced as submitted transactions do not have to be sent as a network request to a hosted blockchain node service which is exposed over API and may have rate limiting.

However, there is still the problem of latency with data propagating across the blockchain network to other nodes, resulting in slow processing of transactions, and the "getting the block late" problem on the data receiving side as described in the previous section.

Running a node also allows access to the Mempool and in future work could be used to predict potential profitable opportunities before they exist on-chain. The author did not run a local node for testing due to the large space requirements which he did not have access to nor the additional capital to pay for storage space on the cloud.

### Optimal Architecture

A problem arises in the previously discussed architectures (Figure 3.1 and 3.2) where the script gets the block late. This occurs when a new block has been added to the blockchain, but it has not propagated over the network yet, and the hosted node the script is connected to is acting on stale data. If a profitable opportunity was found in this case by the bot it would likely fail as some other party would have already submitted a transaction to act upon it.

The optimal architecture for the system would tackle this by running numerous servers, each hosting its own Ethereum node and the trading bots. These would be distributed geographically to ensure they have access to different synchronisations of the blockchain before they come into consensus on the state of the network. This can be visualised below in Figure 3.3.

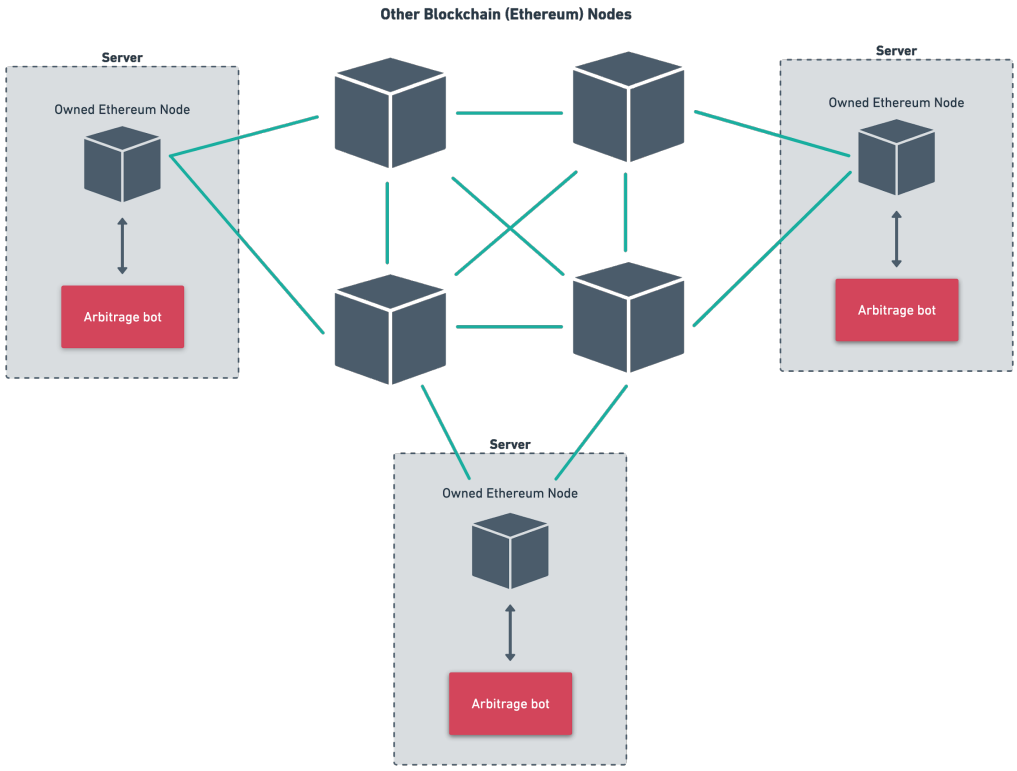


*Figure 3.3: Optimal System Architecture*

These would also have access to different Mempools as when a transaction is submitted to the Mempool, each node broadcasts it peer-to-peer to other nodes. This means that there is an information asymmetry across the network's Mempools also as again information takes time to propagate across nodes. The Mempool is not used in this work however in future works this could be considered.

There is a trade-off here between the cost of running a node on the cloud, the profit the nodes generate from mining, and the additional profit made from the trading bots by distributing nodes. An optimal distribution would require analysis and comparison with the cost trade-offs to be found. This architecture was unfeasible to implement due to capital restraints.

**Problems with Optimal Architecture**

With an optimal architecture implemented, there is still the problem of front running by other bots and by the mining pools themselves. This is discussed by the anonymous (2021) author behind *Rekt,* a highly popular and credible decentralised finance news source, where they state a company known as *BloXroute* provide customers with a “high-speed network” of nodes and provide privileged “access to mining pools, and the ability to bypass the heavily armed bots that patrol the Mempool.” They state 40-50% of all miners who maintain the network are running *BloXroute* nodes which is turning Ethereum into a pay-to-win service. Miners also extract additional MEV using the methods discussed in Chapter 2.6.5 of this work.

**Solutions to Optimal Architecture Problems**

There are two possible solutions which could be considered. The first being the “if you can’t beat them join them” approach. The author's own opinion on whether the front running and MEV extraction techniques and services are ethical has changed (numerous times) throughout the course of his research. The author’s views were initially influenced by the illegality of front running in traditional finance, however as there is no regulation against it in decentralised finance by not using the services a future researcher is at a disadvantage compared to other users who would be using such services. There is an ethical decision to be made here as being ethical in a traditional finance sense by not engaging in these practices is unprofitable, however within a decentralised finance context the ethicality of these practices remains up for question.

As the services exist for private node access which offer the ability to bypass the Mempool and the ability to pay for ordering optimisation, if in future work these are not used in the implementation the researcher would be at a significant disadvantage.

The other solution would involve adapting the trading methods to avoid detection of front running bots. This can be done using some form of obfuscation, which is not widely popular, or by targeting opportunities that the front running bots do not look for or detect. The strategy of targeting these lesser-known opportunities will be described in Chapter 4.5

# Chapter 4 | Technical Implementation

## 4.1 Overview

The implementation phase involved testing numerous arbitrage methods and evaluating the results to examine if there are potentially profitable trading strategies in decentralised finance at the time of writing. The strategies tested were:

- Two altered versions of the buy-low sell-high trading strategy
- Liquidations of under-collateralised loans
- An investigation of the triangular arbitrage method which remains unfinished.

## 4.2 Technology used

### 4.2.1 Core Languages & Runtime Environments

**Ethereum**

Ethereum is an open-source, decentralised blockchain that allows the writing of smart contracts on the blockchain (Frankenfield and Anderson, 2021). Ethereum was chosen by the author due to being the most popular blockchain for transaction volume, and also being the most widely used blockchain for decentralised finance.

**Solidity**

Solidity is an object-oriented, high-level, Turing complete programming language for implementing smart contracts and is designed to target the Ethereum blockchain (Solidity — Solidity 0.7.4 documentation, 2021).

**JavaScript**

JavaScript is a dynamically typed programming language and was chosen due to the multitude of Web 3.0 tools and supporting libraries built for the runtime environment, which was chosen, NodeJS.

**NodeJS**

Node.js is an asynchronous event-driven JavaScript runtime environment. NodeJS was used for writing the scripts that monitor prices and perform calculations before triggering the smart contracts. NodeJS also allows usage of the Node Package Manager (NPM) which allows usage of JavaScript libraries published by other users, which the author took advantage of.

**GraphQL**

GraphQL is a query language for APIs and a runtime for fulfilling queries, an alternative to REST. The author used the GraphQL query language when querying The Graph network for the *Flash Liquidation* Method.

### 4.2.2 Frameworks

#### Truffle

Truffle is a development environment and testing framework for blockchains using the Ethereum Virtual Machine (Truffle Suite, 2021). It offers built-in smart contract compilation and deployment, automated contract testing and package management that integrates with NPM. Truffle was used for development, testing and deploying the contracts.

### 4.2.3 APIs & Services

#### Infura

Infura provides API access to the Ethereum network, supporting both JSON-RPC over HTTPS and WebSocket interfaces, providing request and subscription-based connections. Infura's free tier is used for testing. Latency times with Infura can be problematic as they throttle the accessing of their API, so for submitting transactions on Mainnet the author used a premium blockchain node service.

#### QuikNode

QuikNode provides API access to the Ethereum network through a dedicated personal node, supporting both JSON-RPC over HTTPS and WebSocket interfaces, providing request and subscription-based connections. It is a paid service the author used to ensure low latency times when running the application in production.

**Speed Comparison of Infura and QuikNode**

The author tested his custom Infura endpoint against QuikNodes endpoint using a comparison service. The results shown below in Figure 4.1 indicate a speed increase of over 150%.

| Endpoint | Avg response time | 200s |
|---|---|---|
| Baseline QuikNode API (with launch plan) | 59.6ms | 308 |
| mainnet.infura.io | 154.5ms | 308 |

*Figure 4.1: Speed Comparison of Infura and QuikNode*

*(Web3 Endpoint Benchmarking Tool - QuikNode, 2021)*

**Etherscan**

Etherscan is a free service that allows users to search the Ethereum blockchain for transactions, addresses, token prices and explore blocks. This was used to monitor transactions upon receiving hashes and also for examining potential front runs. Etherscan also offers a service for the Kovan test network which the author also used for examining test transactions.

**CoinGecko**

CoinGecko is a service that provides an analysis of current and historical cryptocurrency markets. It tracks prices, volume and market capitalisation. CoinGecko's API was used to check token prices when calculating trade profitability within some of the implemented methods.

**GasNow**

GasNow is a public API to estimate Ethereum gas price for different speeds of trades based on the number of pending transactions (Gasnow.org, 2021). The API is used to fetch the current gas price processing times, which ensures transactions are always submitted with a high gas price for quick mining.

### 4.2.4 Decentralised Finance Protocols

**Uniswap**

Uniswap is a decentralised exchange (DEX) that is considered an automated market maker (AMM) as it uses a constant product formula model and liquidity pools. Uniswap is used in this work for both arbitrage and swapping tokens when needed. Revisit Chapter 2.4.2 for more information about the technical underpinnings of the protocol. Uniswap also allows for flash loans called "*flash swaps*" which are used for the *Flash-Low Sell-High* trading method.

**Sushiswap**

Sushiswap is a fork of Uniswap's original code base that developers then added additional features to. As Sushiswap's liquidity pools are independent of Uniswap's, price discrepancies often exist between them which are opportunities for profitable trades. They use the same implementation of liquidity pools with the constant product model and nearly identical code, however, they do not support *flash swaps*.

### Aave

Aave is an open-source and non-custodial liquidity protocol for earning interest on deposits and borrowing assets (Aave – Open Source DeFi Protocol, 2021). Aave is used in this work for accessing large amounts of capital through flash loans and for the *Flash Liquidation* trading strategy in which unhealthy loans are liquidated for profit.

### The Graph Protocol

The Graph is an indexing protocol for querying decentralised networks like Ethereum. It allows users to build and publish open APIs, which are called subgraphs, that anyone can then query (The Graph, 2020). It uses the GraphQL query language.

Before The Graph, blockchains were mainly queried over API by building centralised indexing servers, and having these servers pull data from blockchain networks, store them and finally expose them over API (Tal, 2018). There are numerous risks here for users of the API such as the risk of the service failing. The Graph solves this problem by hosting the subgraph indexing nodes as a peer-to-peer network. *Indexers* run the nodes that support the network and *curators* organise data into the subgraphs which can then be easily queried (The Graph, 2020). This creates a much more resilient network with no single source of failure or centralisation risks. The Graph is a complex protocol that will not be covered in detail in this work, however it is used for fetching data within the *Flash Liquidate* implementation.

## 4.3 Flash-Low Sell-High

**Overview**

Two decentralised exchanges were used for this method, Uniswap and Sushiswap and the differences in prices between the pools monitored using a NodeJS script. As the author had limited capital available and the price difference between exchanges is usually small, a few percent at most, for a trade to be profitable when taking account of the transaction fees the trade must be of sufficient size.

To solve this problem the author used the new financial instrument, the flash loan, to access the required capital for the "buy" part of the *buy-low sell-high* method. This flash loan was taken from Uniswap using their *flash swap* feature. Following this, the token was swapped on Sushiswap for the second token at a better rate, before reimbursing the flash loan on Uniswap and sending profits back to the owner's wallet.

This method is exclusive at the time of writing to the Uniswap exchange as it allows reimbursement of loans in either token requiring the constant product model formula is not invalidated, in the sense that the two reserves of tokens $x$ and $y$ must still have the constant product of $k$ upon completion.

### Example of Transaction Flow

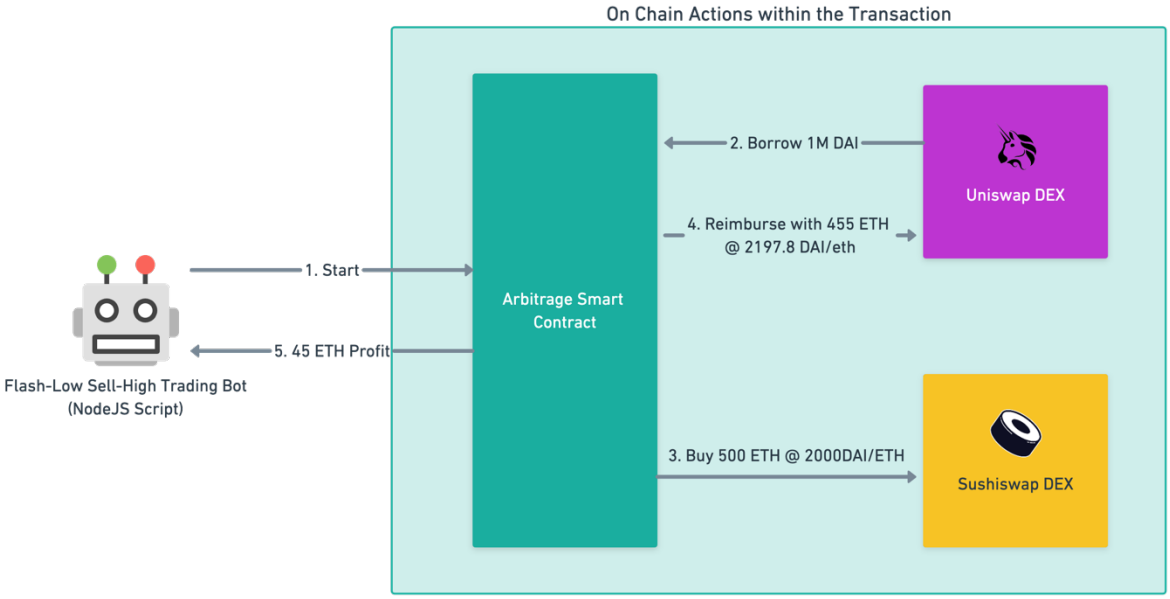


*Figure 4.2: Example Transaction Flow for Flash-Low Sell-High*

### Trading Bot

The author implemented a Node.js script for the trading bot. It commences by monitoring the price difference between both exchanges for the tokens. As each exchange has an independent liquidity pool, the two spot exchange rates are derived from the constant product formula model as both Uniswap and Sushiswap are automated market makers. As described in Chapter 2.4.2 these pools can be represented as $x_1 * y_1 = k_1$ and $x_2 * y_2 = k_2$, where the former represents Uniswaps values and the latter represents Sushiswaps values, and the spot exchange rates as $x_1/y_1$ and $x_2/y_2$. Using the Uniswap SDK, which functions for both Uniswap and Sushiswap as Sushiswap was a fork of Uniswap, functions are called to get the prices on the tokens trading in the pair, represented in terms of the other token in the pool. This allows the monitoring of the difference between the two spot exchange rates.

After the prices and number of each token in the pools are fetched (the reserves), the potential profitability excluding gas fees is calculated by subtracting the liquidity provider (LP) fee. This is 0.3% for each exchange, and as both exchanges are traded through this fee is paid twice. This is deducted from the difference ratio calculated to give a potential profitability after LP fees but excluding gas fees.

```
function estimateProfitAfterTradingFees(uniswapPrice, sushiswapPrice) {
    const diff = Math.abs(sushiswapPrice - uniswapPrice);
    const diffRatio = diff / Math.max(sushiswapPrice, uniswapPrice);

    // subtract fee twice as we pass through pools twice
    const fees = LIQUIDITY_PROVIDER_FEE * 2;

    return diffRatio - fees;
}
```

*Figure 4.3: Function to Find Profitability after LP fees, Represented as a Ratio*

Following this, a profitable trade size accounting for price impact must be determined. Due to the constant product formula $x * y = k$, the size of the trade will affect the exchange rate. To find the optimal trade size the number of starting tokens must be found that gives the most finishing tokens after making the trade and repaying the flash swap. If starting the trade with token $x$, this would be maximising the difference between $\Delta x_2$ and $\Delta x_1$. This can be solved by representing the changes within the constant product model as $(x_1 + \Delta x_1) * (y_1 + \Delta y_1) = k_1$ and $(x_2 + \Delta x_2) * (y_2 + \Delta y_2) = k_2$. These formulae can then be rearranged to find the output after each trade, which is $\Delta y_1$after the first trade using input amount $\Delta x_1$ and $\Delta x_2$ after the second trade using input amount $\Delta y_1$. This corresponds to $\Delta y_1 = \frac{k_1}{x_1 + \Delta x_1} - y_1$ and $\Delta x_2 = \frac{k_2}{y_2 + \Delta y_1} - y_2$. This can then be solved iteratively to optimise for the positive greatest difference between $\Delta x_2$ and $\Delta x_1$, where $\Delta x_1$is the starting amount and $\Delta x_2$ is the finishing amount. The

absolute values are taken from the equations in the form $\Delta y = \frac{k}{x + \Delta x} - y$ as when the swap is made by depositing one token, increasing the $\Delta$, the other token is withdrawn thus decreasing the $\Delta$. This negative $\Delta$ value will be the number of received tokens withdrawn, therefore the absolute value is taken. The *findOptimalTradeAmount* function achieves this and can be seen in Figure 4.4, pseudo code explaining this is also displayed below.

<u>Pseudo Code of Core Optimisation Logic Displayed in Figure 4.4</u>

bestDifference = 0

optimalSize = 0

$\Delta x_1$ = 1

while $\Delta x_1 < x_1$ AND $\Delta x_1 < x_2$

$\Delta y_1 = | \frac{k_1}{x_1 + \Delta x_1} - y_1 |$

$\Delta x_2 = | \frac{k_2}{y_2 + \Delta y_1} - x_2 |$

currentDifference = $\Delta x_2 - \Delta x_1$

if currentDifference > bestDifference

bestDifference = currentDifference

optimalSize = $\Delta x_1$

$\Delta x_1 = \Delta x_1 + 1$

```
function findOptimalTradeAmount(uniswapReserves,sushiswapReserves,token){
    let x1 = null
    let y1 = null
    let k1 = null
    let x2 = null
    let y2 = null
    let k2 = null

    if(token == 1){ //uni > sushi start on token 1 (x)
        x1 = uniswapReserves['token1']
        y1 = uniswapReserves['token2']
        k1 = x1 * y1
        x2 = sushiswapReserves['token1']
        y2 = sushiswapReserves['token2']
        k2 = x2 * y2
    }

    if(token == 2){ // sushi > uni start on token 2 (x)
        x1 = uniswapReserves['token2']
        y1 = uniswapReserves['token1']
        k1 = x1 * y1
        x2 = sushiswapReserves['token2']
        y2 = sushiswapReserves['token1']
        k2 = x2 * y2
    }

    let increment = 1
    let startAmount = 1
    if(x1  < 1000 || x2 < 1000){ // if pool units are smaller use smaller values
        increment = 0.01
        startAmount = 0.01
    }

    let optimalTradeSize = 0;
    let bestDifference = 0;

    while(startAmount < x1 && startAmount < x2){

        let outputY = Math.abs((k1/(x1+startAmount)) - y1)
        let outputX = Math.abs((k2/(y2+outputY)) - x2)
        let difference = outputX - startAmount
        if(difference < 0) break;
        if(difference > bestDifference){
            bestDifference = difference
            optimalTradeSize = startAmount
        }
        startAmount = startAmount + increment
    }

    return {
        optimalTradeSize,
        bestDifference
    }
}
```

*Figure 4.4: Finding Optimal Trade Size*

After calculating the optimal trade size and returning the difference profited in token units, the profit in token units is then converted to fiat before deducting transaction costs to give a net profit value. As this trade is likely to be front run, a fast gas fee is used to ensure it is mined as quickly as possible, reducing the amount of time front runners have to copy the trade. To achieve this, an API was used, GasNow, to get the fastest gas of the transaction in the previous block, plus an additional small amount in case of variance. If this final profit value is positive, the smart contract is called using the function displayed in Figure 4.5. This instantiates the contract object using its application binary interface (ABI) and the tokens address for the correct network being used. Then an array of the arguments is created consisting of the token addresses and the size of the flash loan, before calling two further functions. One to create a transaction using the arguments and another to sign and send the transaction before returning the result.

```
async function executeTrade(token0Address, token1Address, amount0, amount1,from,
privateKey, web3, network){

    let arbitrageContractAddress = (network === 1 ?
process.env.ARBITRAGE_CONTRACT_MAINNET : process.env.ARBITRAGE_CONTRACT_KOVAN)

    let args = [token0Address,token1Address,amount0,amount1]

    try{
        let arbitrageContract = new web3.eth.Contract(ARBITRAGE_CONTRACT_ABI,
arbitrageContractAddress)
        let tx = await createTx(arbitrageContract,
'startArbitrage',from,arbitrageContractAddress,args,web3)
        let result = await signAndSendTransaction(tx, privateKey, web3)
        return result
    }catch(e){
        return e
    }
}
```

*Figure 4.5: Function Called to Trigger Flash-low Sell-High Smart Contract to Execute Trade*

The *createTx* function, displayed in Figure 4.6 and called in Figure 4.5, creates a raw Ethereum transaction by using the specific contract needed for the trading method, the specific method called in the contract to start the trading sequence, and the arguments to call it. This function is used in all methods.

```
 async function createTx(contract, method, from, to, args, web3){
    const methodCall = contract['methods'][method]

    const gasPrice = Number(await getFastestGasPriceWei());
    const gas = "1000000" // Max amount gas used - this will never be reached
    const tx = {
        from: from,
        to: to,
        data: methodCall.apply(null,args).encodeABI(),
        gas,
        gasPrice,
        gasLimit: gas * gasPrice,
    };
    return tx;
}
```

*Figure 4.6: Function to Create a Raw Transaction*

The transaction is then signed using the private key of the author which is stored in an environment variable for security reasons and passed down through previous function calls.

```
async function signAndSendTransaction(tx, privateKey, web3){
    const signedTx = await web3.eth.accounts.signTransaction(tx, privateKey);
    return web3.eth.sendSignedTransaction(signedTx.raw ||
signedTx.rawTransaction)
}
```

*Figure 4.7: Function to Sign Transactions Using Private Key*

This method could be improved by also accounting for a percentage of transactions which are front run, however this could not be derived without testing. This will be discussed in the evaluation section in Chapter 5.6.

All potentially profitable opportunities stored and saved to file upon termination of the script where they can be analysed and visualised using scripts developed by the author. In the case of running in production results of executed trades are also stored.

**Smart Contract Structure**

The *Flash-Low Sell-High* smart contract consists of two main functions, one which is triggered to start the arbitrage by the Node.js script. This gets a flash loan for one of the tokens from Uniswap and the other which is triggered by a call-back after the flash loan has been received from Uniswap, which then exchanges the token on Sushiswap for the second token before paying back the flash loan to Uniswap and transferring any profits remaining to the account which initiated the sequence.

```
function startTradingSequence(
    address token0,
    address token1,
    uint amount0,
    uint amount1
  ) external {

    // only let owner call the contract - may protect against some frontrunning
    require(msg.sender == owner, "!owner");

    address pairAddress = IUniswapV2Factory(uniswapFactory).getPair(token0,
token1);

    // ensures pair exists on UNI
    require(pairAddress != address(0), '!poolExists');

    // initiate flash loan (flash swap)
    IUniswapV2Pair(pairAddress).swap(amount0, amount1, address(this), bytes('not
empty'));
  }
```

*Figure 4.8: Flash-Low Sell-High Smart Contract Function Called to Start Trade*

As seen in Figure 4.8, this contract is restricted so that only the deployer of the contract can call the *startTradingSequence* function. This provides some protection against front running as another user could not replicate the submitted trade using the same contract as the author. Generalised bots who may already have a similar contract deployed will still be a threat.

```
function uniswapV2Call(address _sender, uint _amount0, uint _amount1, bytes
calldata _data) external {

    // get tokens from unipair pool
    address token0 = IUniswapV2Pair(msg.sender).token0();
    address token1 = IUniswapV2Pair(msg.sender).token1();

    // Ensures comes from UNI pair contract
    require(msg.sender == UniswapV2Library.pairFor(uniswapFactory, token0,
token1), '!authorised');

   // get amount of token borrowed
    uint amountToken = _amount0 == 0 ? _amount1 : _amount0;

    address[] memory path = new address[](2);

    // define the direction of the trade
    path[0] = _amount0 == 0 ? token1 : token0;
    path[1] = _amount0 == 0 ? token0 : token1;

    // pointer to token to sell on sushi
    IERC20 token = IERC20(_amount0 == 0 ? token1 : token0);

    // aprove the contract
    token.approve(address(sushiRouter), amountToken);

    // calculate amount need to reimburse Uniswap
    uint amountRequired = UniswapV2Library.getAmountsIn(factory, amountToken,
path)[0];

    // sell the token we borrowed on uniswap to sushi
    uint amountReceived = sushiRouter.swapExactTokensForTokens(amountToken,
amountRequired, path, address(this), timeLimit)[1];

    // get pointer to output token
    IERC20 otherToken = IERC20(_amount0 == 0 ? token0 : token1);

    // reimburse uniswap amount
    otherToken.transfer(msg.sender, amountRequired);

    // send profit back to address who initiated transaction (script address
that calls it)
    otherToken.transfer(tx.origin, amountReceived - amountRequired);
  }
```

*Figure 4.9: Flash-Low Sell-high Smart Contract Function Triggered on Callback from Uniswap*

**Theoretical Results**

The high gas fees on Ethereum at the time of testing introduced a financial risk as if the transactions were to be unsuccessful and fail, they would still incur a cost. The author decided to run the full script for 100 iterations across a number of popular pairs which were active on both Uniswap and Sushiswap to get a theoretical result of how profitable the bot may be, without executing the trades. This took approximately 2 hours. It is worth noting that the author used the free service Infura here which is significantly slower than QuikNode. The author wanted to save his limited QuikNode credits for production usage and executing the trades to determine the effect of front running on live transactions. The author wanted to calculate the theoretical profit of the bot if it were to execute every profitable trading opportunity it found to gain insights into potential profitability before deploying the contracts. The results of this are displayed below in Table 1.

| **Trade Number** | **Token Pair (A/B)** | **Trade Size (Token)** | **Uniswap Price (A/B)** | **Sushi Swap Price (A/B)** | **Profit before Gas Fee ($)** | **Net Profit ($)** |
|---|---|---|---|---|---|---|
| 27 | BADGER/WBTC | 0.38 WBTC | 16781.21 | 1666.05 | 78.72 | 23.49 |
| 26 | WETH/VSP | 390 VSP | 0.0130431 | 0.0129398 | 55.96 | 4.54 |
| 25 | WETH/ REN | 10432 REN | 0.000346505 | 0.000343398 | 44.99 | 5.27 |
| 24 | WETH/VSP | 393 VSP | 0.0130449 | 0.012938 | 56.81 | 15.23 |
| 23 | WETH /BAO | 8703153 BAO | 0.000000375 341 | 0.000000369 581 | 69.24 | 14.09 |
| 22 | WETH/NFTX | 28 NFTX | 0.0489485 | 0.0473794 | 65.03 | 4.92 |
| 21 | BADGER/WBTC | 692 BADGER | 1632.49 | 1645.57 | 94.69 | 43.35 |

*Table 1: Snippet of Flash-Low Sell-High Theoretical Results (27 - 21)*

| | |
|---|---|
| **Iterations** | 100 |
| **Unique Opportunities** | 27 |
| **Gross Profit** | $2491.28 |
| **Net Profit** | $1150.89 |

*Table 2: Summary Flash-Low Sell-High of Theoretical Results*

As can be seen in Table 2, the developed bot had a theoretical net profit of $1150.89 over the period, if all of the transactions were to be successfully completed without being front run or failing as a result of being beaten to the opportunity by another trader. The distribution of the trades when looking at how many blocks each lasted is worth examining.

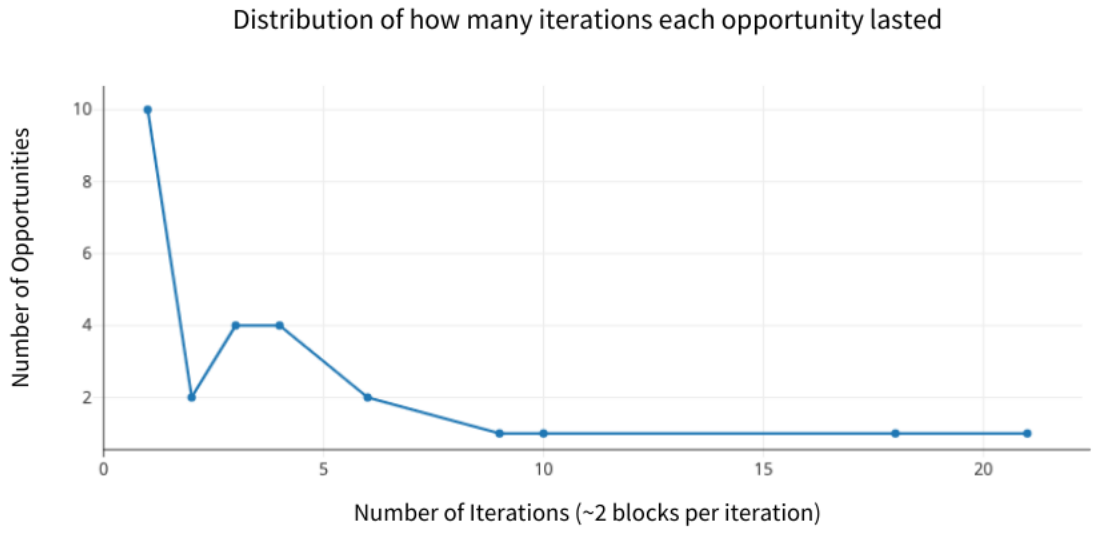


*Figure 4.10: Graph Showing How many Iterations Trading Opportunities Lasted*

As can be seen in Figure 4.10, ten of the opportunities existed for only a single iteration (37%) and 20 (27%) existed for less than 5 iterations with few outliers lasting longer.

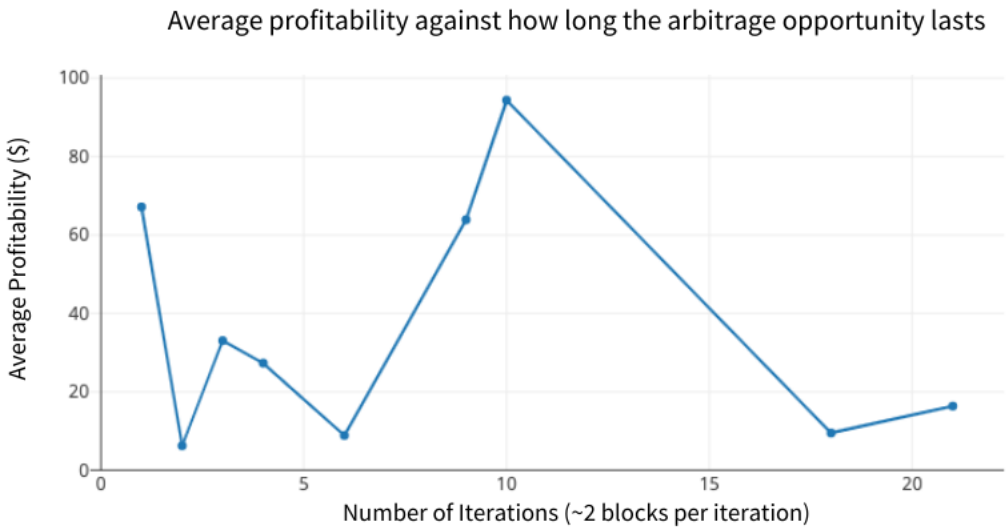


*Figure 4.11: Graph of Average Profit against Number of Iterations Opportunities Lasted*

As can be seen in Figure 4.11, the opportunities that lasted only a single iteration were significantly more profitable (average $74) than the other opportunities lasting under 5 iterations. The most profitable opportunity was also in this group. When looking at the rest of the data points, the opportunities existing for over 5 iterations, a large spike can be seen indicating a large average profitability for the group at the peak of the datapoint. However, after 5 iterations the opportunities only had a single data point, it is likely these were outliers and do not occur frequently. These opportunities may have been turned into gas auctions as mentioned in Chapter 2.6.5 where the gas is slowly bid up by competing bots.

**Actual Results of Executing Transactions**

The author ran the bot in production mode using QuikNode for increased speed, to test it on the Ethereum Mainnet. As each transaction has a fee of about $90 the author had limited capital to test and needed to hit a profitable trade within the first few transactions Out of the first 5 transactions all 5 failed, two due to deflationary *tokenomics* (economics of a token) where tokens are burned as they are sent which led to trade size calculations being incorrect, and three due to being too slow or by being front run. An example of the latter will now be shown and evaluated.

The author's trading bot detected an arbitrage opportunity between SNX/WETH on the two exchanges. The spot exchange rate of SNX/WETH on Uniswap was 0.0108864 and on Sushiswap it was 0.0110268 the failed transaction can be seen in Figure 4.12.

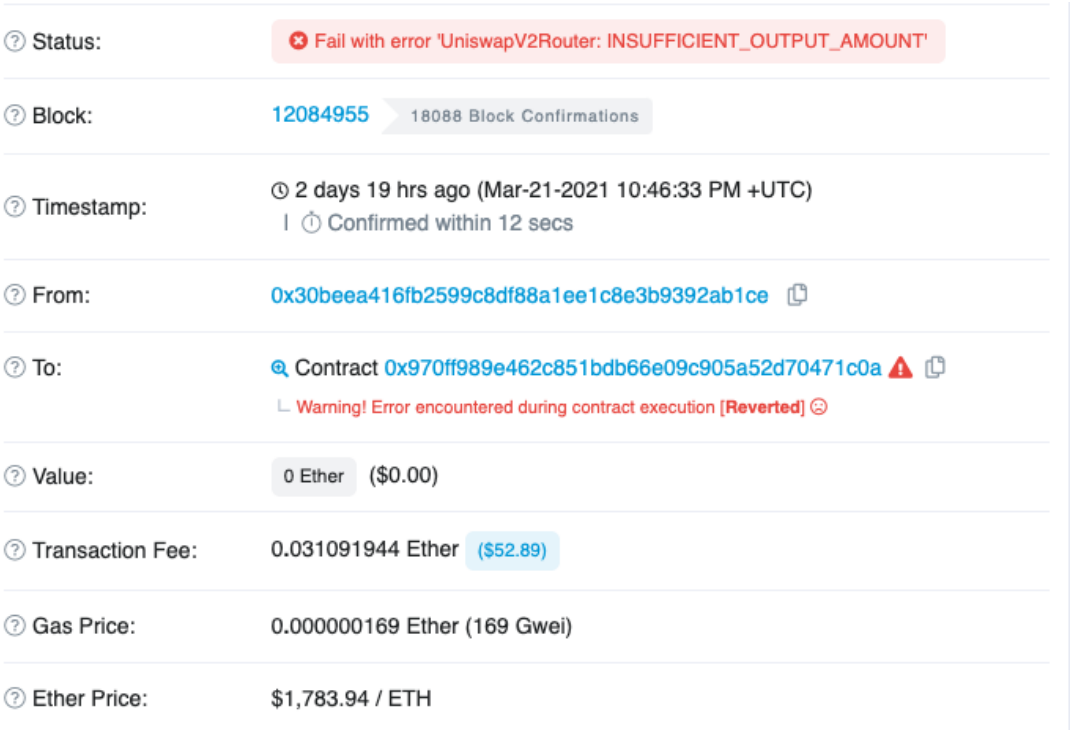


*Figure 4.12: Failed Flash-Low Sell-High Transaction due to Exchange Rate Changing*

The author familiarised himself with the Uniswap protocol using their Github as the code is open source and investigated the error. This error is caused when there are insufficient funds after one of the trades is requested, which is due to the exchange rate changing before the transaction was confirmed. The author investigated Etherscan and filtered by transactions on the SNX/WETH pair. The author found a trade placed before his transaction acting on the same arbitrage opportunity.

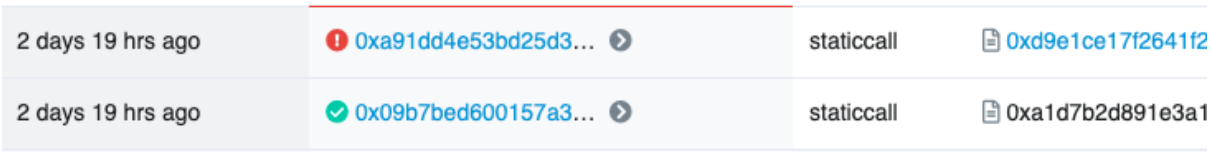


*Figure 4.13: The Authors Transaction and the Successful Transaction*

The author then examined this transaction, displayed in Figure 4.13 with the hash beginning with 0x09b7. The transaction bought SNX on Uniswap for WETH before selling it on Sushiswap. This happened within a single transaction, implementing a standard *buy-low sell-high* strategy within the smart contract it called. It is worth noting the timestamps of the transactions here. The transaction that beat the authors transaction to the opportunity was submitted 30 seconds earlier. This is likely due to network latency as the naive architecture was implemented that was explained in Chapter 3.5 and not the optimal architecture, or a lack of optimisation for speed in the author's script, rather than a front run.

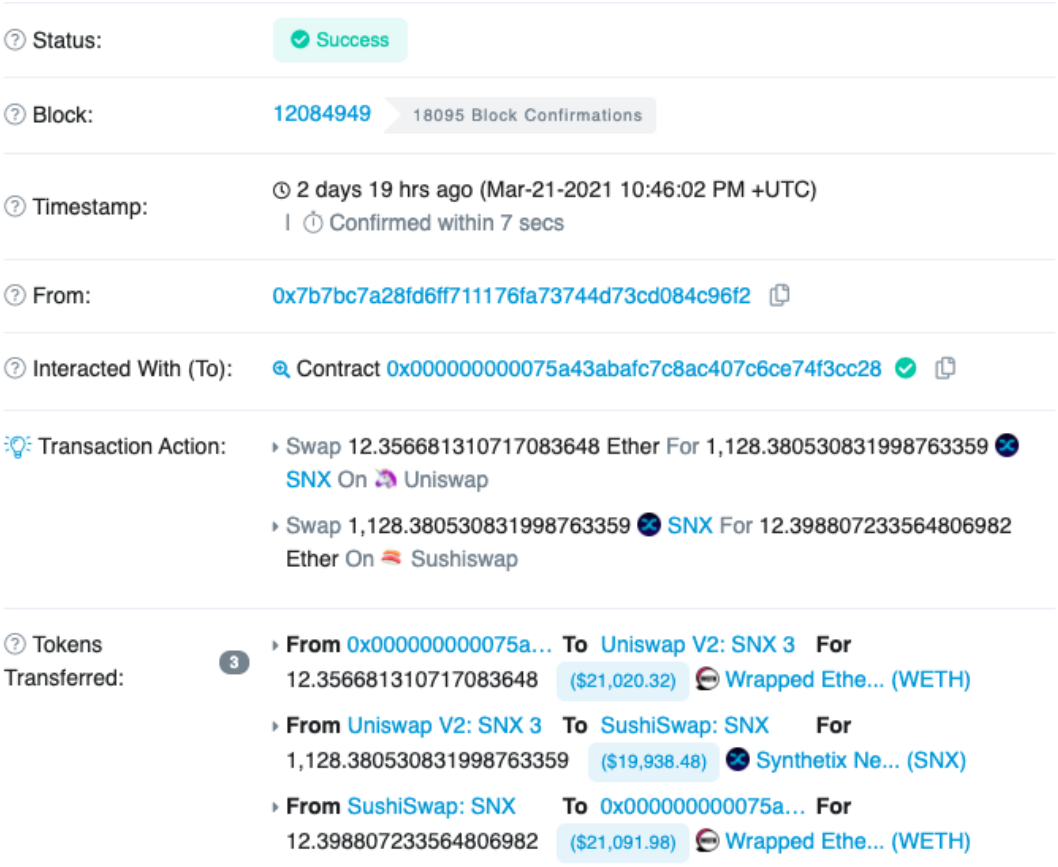


*Figure 4.14: The Successful Transaction*

Although the author had limited capital available and thus a small sample size of transactions to evaluate on Mainnet, the competitiveness of the trading and arbitrage space is common knowledge within the decentralised finance community and is clearly demonstrated in the results of this section. Following this, the author devised a strategy to apply a similar method as described in this Chapter to different exchanges with the hopes of less competition for opportunities. This will be discussed in the *Flash Buy-Low Sell-High* section of this work.

## 4.4 Flash Liquidation

### Overview

The liquidation bot monitors loans taken out by users on Aave using The Graph protocol. When unhealthy loans are found, it then calculates if the loan would be profitable to liquidate before calling the smart contract to trigger a liquidation using a flash loan to access the needed capital. To give an example of how this process works, take one party, Bob, who deposits 10 ETH and borrows 5 ETH worth of DAI in the Aave protocol. If the value of ETH on the open market falls relative to DAI and results in Bob's health factor dropping below 1 his loan will be eligible for liquidation. A liquidator can then repay up to 50% of a single borrowed amount which is 2.5 ETH worth of DAI. In return, the liquidator can claim the collateral which is ETH (and a 5% bonus). The liquidator claims 2.5 ETH (collateral * amount of collateral that can be liquidated) + 0.125 ETH (bonus) for repaying 2.5 ETH worth of DAI (Liquidations, 2021), earning a risk-free profit of 0.125 ETH. As a flash loan is used to access the capital for the liquidation, the flash loan must be reimbursed in the original token the loan was taken out with, this would be DAI in the given example. In this case this would leave the liquidator contract holding ETH after the liquidation. This must then be swapped back to the token the loan was taken out with in order to reimburse the loan. The decentralised exchange Uniswap is used to do this. The collateral token liquidated is swapped back to the initial token, before paying back the loan and returning any remaining profits to the wallet address which initiated the transaction.

**System Architecture including The Graph**

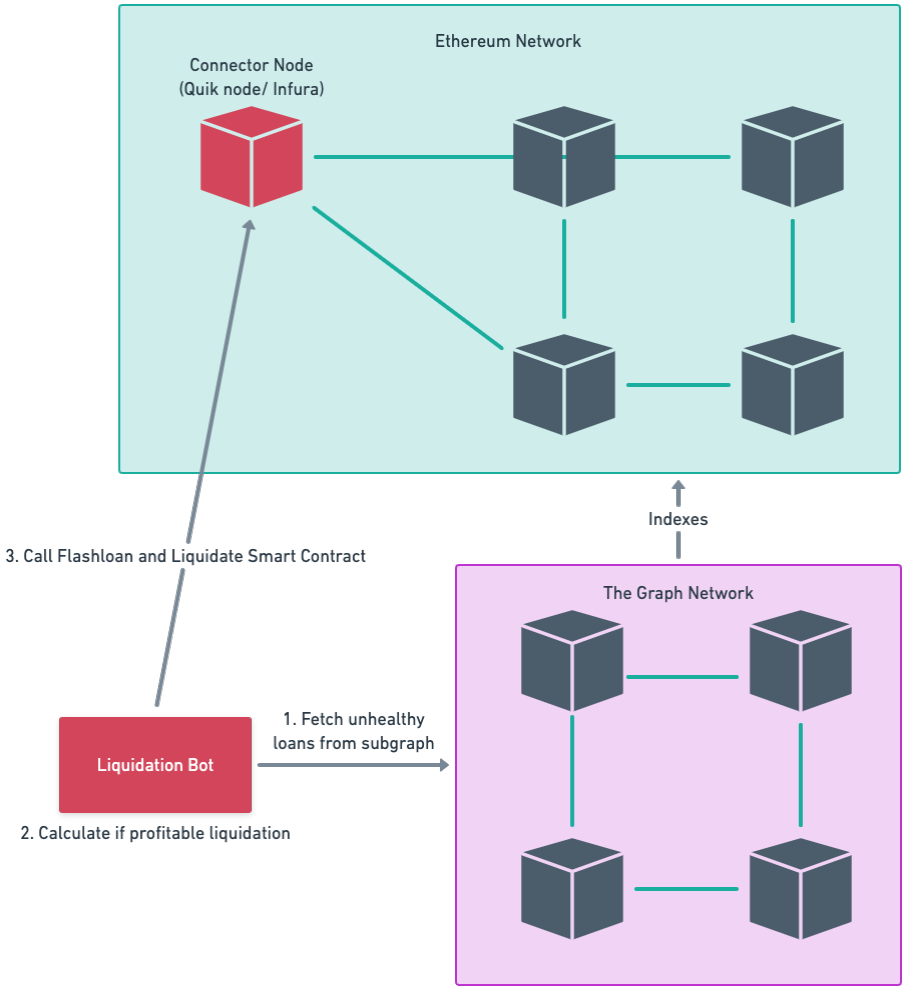


*Figure 4.15: Architecture of the Flash Liquidation System*

The author initially planned to use Aave's API to fetch users and find the unhealthy loans, however, at the time of development the API was being reworked and was unavailable. This is a clear demonstration of centralisation failure. Due to this, the author decided to use a decentralised protocol, known as The Graph to fetch the unhealthy loans. This can be seen in the architecture diagram above in Figure 4.15.

**Example Transaction Flow of opportunity found by Liquidator Bot**

One such example of an opportunity found by the liquidator bot is detailed below in Figure 4.16. If the transaction were executed successfully and not front run the transaction flow would be as follows.

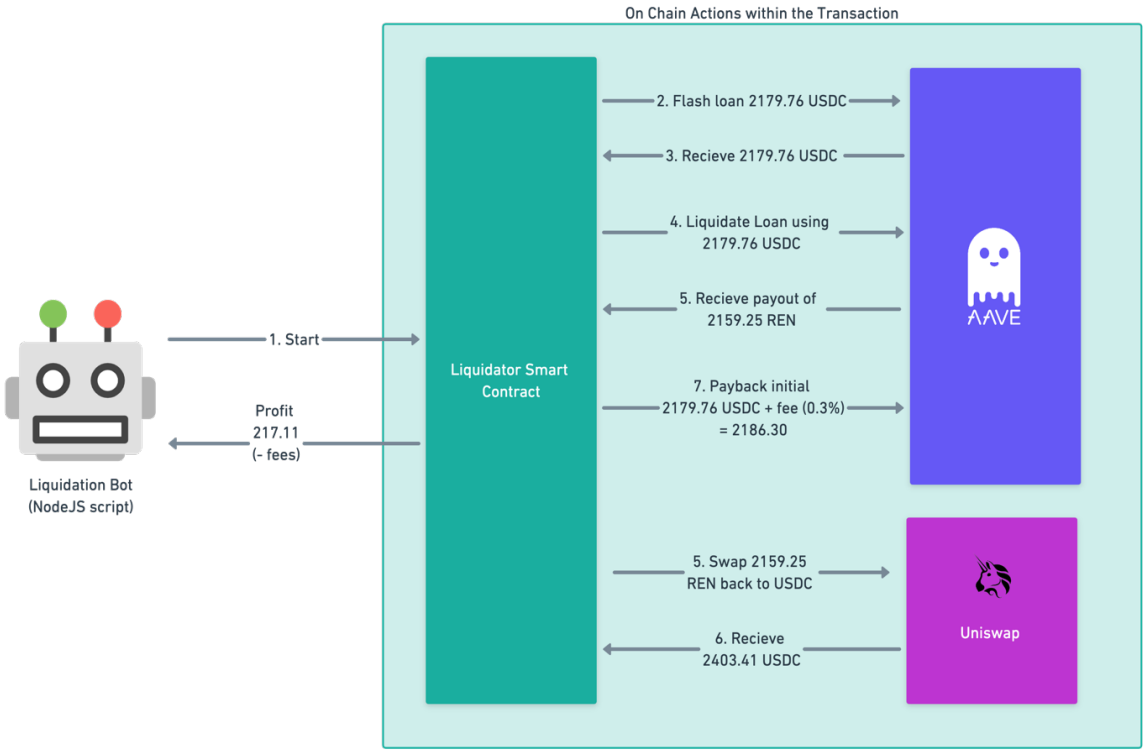


*Figure 4.16: A Profitable Liquidation Visualised, found by the Authors Liquidator Bot*

### Liquidator Bot

The author built a Node.js script for the trading bot. The bot first queries Aave users from The Graph Protocol by querying the Aave subgraph. This is done using the GraphQL query language. The query is also paginated and returns 1000 loans at a time, as seen in Figure 4.17.

```
 query GET_LOANS {
        users(first:1000, skip:${1000*count}, orderBy: id, orderDirection: desc,
where: {borrowedReservesCount_gt: 0}) {
          id
          borrowedReservesCount
          collateralReserve:reserves(where: {currentATokenBalance_gt: 0}) {
            currentATokenBalance
            reserve{
              usageAsCollateralEnabled
              reserveLiquidationThreshold
              reserveLiquidationBonus
              borrowingEnabled
              utilizationRate
              symbol
              underlyingAsset
              price {
                  priceInEth
                }
              decimals
            }
          }
          borrowReserve: reserves(where: {currentTotalDebt_gt: 0}) {
            currentTotalDebt
            reserve{
              usageAsCollateralEnabled
              reserveLiquidationThreshold
              borrowingEnabled
              utilizationRate
              symbol
              underlyingAsset
              price {
                priceInEth
              }
              decimals
            }
          }
        }
      }
```

*Figure 4.17: GraphQL Query to Fetch Loans from the Aave Subgraph on The Graph Network*

After the data is fetched from the Aave subgraph, a calculation is made for each user to find their health factor. A user's health factor determines if their loans can be liquidated. A user's loan is defined as unhealthy when their health factor drops below 1.0. This happens when the collateral decreases in value or borrowed debt increases in value (Liquidations, 2021). The health factor can be formally described as such, where if $H_f < 1$, the user's loan can be liquidated. This would occur when the value of the collateral tokens they have deposited in the protocol to collateralise the loan significantly falls in value relative to the borrowed tokens.

$$H_f = \frac{\Sigma\ Collateral_i\ in\ ETH\ *\ Liquidation\ Threshold_i}{Total\ Borrowed\ in\ ETH\ +\ Total\ Fees\ in\ ETH}$$

The largest loan is also found when computing the values needed to find the health factor. This ensures the script is liquidating the loan with the largest potential return for the user. For each user this process is repeated, and the collection of large unhealthy loans are returned upon completion of the function. This can be seen in Figure 4.18.

```
function getUnhealthyLoans(loanData) {

    let unhealthyLoans=[];

    for(user of loanData.users){
      // initialise needed variables
      let totalBorrowed=0;
      let totalCollateral=0;
      let totalCollateralThreshold=0;
      let maxBorrowedSymbol;
      let maxBorrowedPrincipal=0;
      let maxBorrowedPriceInEth = 0;
      let maxCollateralSymbol;
      let maxCollateralBonus=0;
      let maxCollateralPriceInEth = 0;

      // iterate over users borrowings for each loan
      for(borrowReserve of user.borrowReserve){
        totalBorrowed += borrowReserve.reserve.price.priceInEth *
borrowReserve.currentTotalDebt / (10**borrowReserve.reserve.decimals)
        if (borrowReserve.currentTotalDebt > maxBorrowedPrincipal){
          maxBorrowedSymbol = borrowReserve.reserve.symbol
          maxBorrowedPrincipal = borrowReserve.currentTotalDebt
          maxBorrowedPriceInEth = borrowReserve.reserve.price.priceInEth
        }
      }

      // iterate over users collateral for each loan
      for(collateralReserve of user.collateralReserve){
        totalCollateral += collateralReserve.reserve.price.priceInEth *
collateralReserve.currentATokenBalance /
(10**collateralReserve.reserve.decimals)
        totalCollateralThreshold += collateralReserve.reserve.price.priceInEth *
collateralReserve.currentATokenBalance *
(collateralReserve.reserve.reserveLiquidationThreshold/10000)/
(10**collateralReserve.reserve.decimals)

        if (collateralReserve.reserve.reserveLiquidationBonus >
maxCollateralBonus){
          maxCollateralSymbol = collateralReserve.reserve.symbol
          maxCollateralBonus=collateralReserve.reserve.reserveLiquidationBonus
          maxCollateralPriceInEth = collateralReserve.reserve.price.priceInEth
        }
      }

      let healthFactor = totalCollateralThreshold / totalBorrowed;

      if (healthFactor <= MAX_HEALTH_FACTOR) {
        unhealthyLoans.push( {
            userId  :  user.id,
            healthFactor   :  healthFactor,
```

```
            maxCollateralSymbol : maxCollateralSymbol,
            maxBorrowedSymbol : maxBorrowedSymbol,
            maxBorrowedPrincipal : maxBorrowedPrincipal,
            maxBorrowedPriceInEth : maxBorrowedPriceInEth,
            maxCollateralBonus : maxCollateralBonus/10000,
            maxCollateralPriceInEth : maxCollateralPriceInEth
          })
      }
    }
    // filter out tokens we don't have data for in config
    unhealthyLoans = unhealthyLoans.filter(loan =>
TOKEN_LIST[loan.maxBorrowedSymbol] != null &&
TOKEN_LIST[loan.maxCollateralSymbol] != null)

    return unhealthyLoans;
}
```

*Figure 4.18: Function to Compute Unhealthy Loans from Data Fetched from The Graph*

When these large and unhealthy loans are returned, it is then calculated if each would be profitable to liquidate before triggering the smart contract to start the flash liquidation sequence. First the size of the collateral is taken into account. As specified on the Aave documentation, only a portion of the loan can be liquidated known as the close factor, so the collateral of the loan is multiplied by this close factor. Then the cost of the flash loan is calculated, which is 0.09% for Aave. Following this, the number of tokens liquidated is calculated and converted to Ether for easy comparison. Ether was used over USD for this implementation as the Aave Subgraph stores many of the values in both token units and Ether value which allows for easy calculations without converting to fiat. Then the collateral tokens received by liquidating the loan is calculated. From here we now need to convert back to the initial token the loan was taken out with to reimburse, so the trade back must ensure enough tokens are received after the swap. The author implemented code from Uniswap's GitHub that provided a *useTradeExactIn* method to achieve this, that allows a token amount object to be passed containing the collateral

symbol and the collateral tokens we receive, along with the symbol of the initial token, which then calculates the number of tokens that can be expected to be received by following the optimal trade path that is returned.

A function is then called to get the fastest gas price per unit using the same API as in the previous section which is then multiplied by the amount of gas used to give the gas fee. The amount of gas used was initially found by testing on test networks as after a successful flash liquidation occurred, as seen in Figure 4.23, the amount of gas used could be extracted for future usage. The profit in the borrowed currency after the liquidation is then calculated and converted to Ether to give a profit before gas figure. Gas is then deducted to give a net profit figure. This function then returns the loan to liquidate if it is profitable to do so.

The function displayed in Figure 4.19 is then called to execute upon the opportunity by creating a raw transaction and broadcasting it to the network to initiate the flash liquidation trading sequence in the deployed smart contract.

```
async function executeLiquidation(
  liquidatableTokenAddress,
  collateralTokenAddress,
  flashLoanAmount,
  userAddressToLiquidate,
  amountOutMinimumUniswap,
  swapPathUniswap,
  publicKey,
  privateKey,
  web3,
  network
){

  let liquidatorContractAddress = (network === 1 ?
process.env.LIQUIDATE_CONTRACT_MAINNET : process.env.LIQUIDATE_CONTRACT_KOVAN)

  let args =
[liquidatableTokenAddress,collateralTokenAddress,flashLoanAmount,userAddressToLi
quidate,amountOutMinimumUniswap,swapPathUniswap]
  try{
      let liquidatorContract = new web3.eth.Contract(LIQUIDATOR_CONTRACT_ABI,
liquidatorContractAddress)
      let tx = await createTx(liquidatorContract,
'executeFlashLiquidate',publicKey,liquidatorContractAddress,args,web3)
      let result = await signAndSendTransaction(tx, privateKey, web3)
      return result
  }catch(e){
      // console.log(e)
      return e
  }
}
```

*Figure 4.19: Function Called to Trigger Flash Liquidation Smart Contract*

All potentially profitable opportunities are stored and saved to file upon termination of the script where they can be analysed or visualised, in the case of running in production results of executed trades are also stored.

**Smart Contract Structure**

The smart contract consists of three main functions. The first of which is called by the NodeJS script to get the flash loan and trigger the liquidation sequence.

```
function executeFlashLiquidate(
        address _liquidatableTokenAddress,
        address _collateralTokenAddress,
        uint256 _flashLoanAmount,
        address _userAddressToLiquidate,
        uint256 _amountOutMinimumUniswap,
        address[] memory _swapPathUniswap
    ) public  {

        require(msg.sender == owner, "!owner");

       // params passed to be used in callback
        bytes memory params = abi.encode(
            _collateralTokenAddress,
            _userAddressToLiquidate,
            _amountOutMinimumUniswap,
            _swapPathUniswap
        );
        // the token to be flashed
        address[] memory assets = new address[](1);
        assets[0] = _liquidatableTokenAddress;

        // the amount to be flashed for asset
        uint256[] memory amounts = new uint256[](1);
        amounts[0] = _flashLoanAmount;

        uint256[] memory modes = new uint256[](1);
        modes[0] = 0;

        _lendingPool.flashLoan(
            address(this),
            assets,
            amounts,
            modes,
            address(this),
            params,
            0
        );
    }
```

*Figure 4.20: Smart Contract Function Called to get Flash Loan and Trigger Liquidation Sequence*

Following the successful completion of this function, a call-back function is automatically triggered after the contract receives the flash loan from Aave. This function handles the bulk of the work done within the transaction.

```
function executeOperation(
        address[] calldata assets,
        uint256[] calldata amounts,
        uint256[] calldata premiums,
        address initiator,
        bytes calldata params
    )
        external
        override
        returns (bool)
    {

        (address collateralTokenAddress, address userAddressToLiquidate, uint256
amountOutMinimumUniswap, address[] memory swapPathUniswap) = abi.decode(params,
(address, address, uint256, address[]));

        liquidateLoan(collateralTokenAddress, assets[0], userAddressToLiquidate,
amounts[0], false);

        // Swap token acquired from liquidation (collateral) back to token we
took loan out in.
        swapToBorrowedToken(collateralTokenAddress,amountOutMinimumUniswap,
swapPathUniswap);

        // Calculate Profit
        uint256 profit =
calculateProfits(IERC20(assets[0]).balanceOf(address(this)),amounts[0],premiums[
0]);

        require(profit > 0 , "!Profitable");
        IERC20(assets[0]).transfer(owner, profit);

        // Approve the Aave contract for repayment
        uint amountOwed = amounts[0].add(premiums[0]);
        IERC20(assets[0]).approve(address(_lendingPool), amountOwed );

        return true;
    }
```

*Figure 4.21: Smart Contract Function Triggered after Receiving the Flash Loan from Aave*

It first decodes the parameters passed in the initial function that is triggered to start the trading sequence and receive the flash loan. Following this it liquidates the unhealthy loan of the specified user; this is a simple function which calls the *liquidationCall* method with the required parameters for Aave's lending pool contract. If this is successful, the pool then transfers the collateral tokens to the contract which includes the liquidation bonus as a pay-out. Now the collateral tokens must be swapped to the currency the flash loan was taken out with to reimburse the loan. This is done using Uniswap and the path that was found in the script and passed to the contract as an argument when calling it. This function can be seen in Figure 4.22.

```
function swapToBorrowedToken(
        address startingToken,
        uint amountOutMinimumUniswap,
        address[]
        memory swapPathUniswap
    ) public {

        uint256  amountToTrade = IERC20(startingToken).balanceOf(address(this));
        uint deadline = block.timestamp + 150; // 2.5 mins

        // approve uniswap contract
        IERC20(startingToken).approve(address(uniswapV2Router), amountToTrade);

        // Swaps to required token passed in path path (eg UNI -> WETH -> USDC)
        try uniswapV2Router.swapExactTokensForTokens(
            amountToTrade,
            amountOutMinimumUniswap,
            swapPathUniswap,
            address(this),
            deadline
        ){
        }
        catch Error(string memory reason)
        {
            emit ErrorHandled(reason);
        }
    }
```

*Figure 4.22: Using Uniswap to Swap Token Liquidated to Token Required to Pay Back Flash Loan*

Then profits are calculated by subtracting the loan amount and the fee for the loan from the balance of the token remaining in the account. If the remaining value is above 0, it means after the loan is repaid there is some profit remaining. If this is the case the profit is then transferred to the owner of the contract, before repaying the remaining balance which is the flash loan amount plus the fee to Aave.

**Successful Flash Liquidation on Test Network**

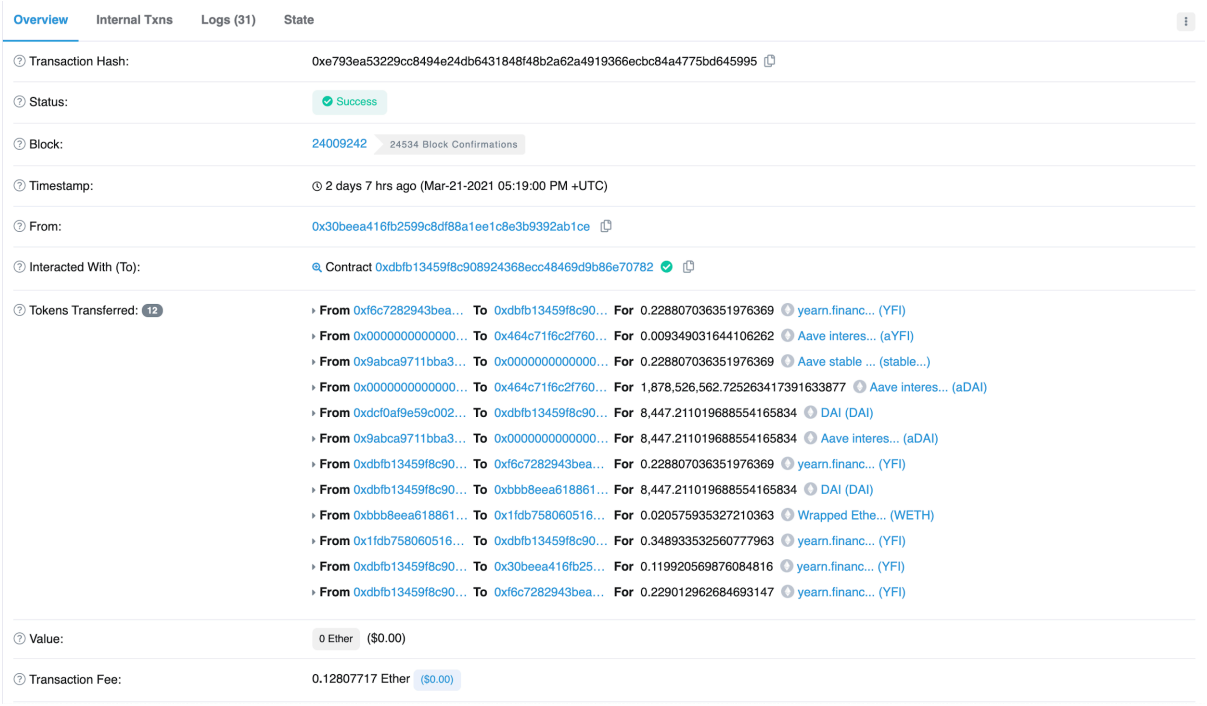


*Figure 4.23: A Successful Flash Liquidation Using a Flash Loan*

The transaction hash of the successful flash liquidation shown above is 0xe793ea53229cc8494e24db6431848f48b2a62a4919366ecbc84a4775bd645995 if the reader would like to examine it on Kovan Etherscan (https://kovan.etherscan.io/).

**Theoretical Results on Mainnet**

The author decided to run the Flash Liquidation script for 1000 iterations to get a theoretical estimation of how profitable the bot may be by liquidating loans. This took approximately 12 hours. It is worth noting that the author used the free service Infura again here which is significantly slower than QuikNode. The results are displayed below in Table 3.

| Block Number | Flash Loan Token | Collateral Token | Best Return Trade | Gas Fee ($) | Net Profit (Ether) | Net Profit ($) |
|---|---|---|---|---|---|---|
| 12238025 | BAT | LINK | LINK -> WETH -> BAT | $150 | 0.091 | 210 |

*Table 3: Flash Liquidation Theoretical Results*

As we can see in Table 3. only a single unique loan was found that could be liquidated for a profit over this period. This loan was detected 6 separate times over the space of 243 blocks, likely meaning that it was liquidated numerous times as the value of the underlying tokens changed causing the users health factor to become unhealthy. It could also be the case gas fee fluctuations meant it was unprofitable to liquidate in some blocks, but profitable in others.

The theoretical results here may not look promising, however at the time of gathering this data volatility was low and most tokens were moderately appreciating across the market. The strategy of liquidating unhealthy loans would generally perform better during periods of uncertainty and in market downturns as discussed in Chapter 2.4.3. The market conditions at the time of testing were not right for this strategy to succeed even theoretically.

**Actual Results on Mainnet**

The author was unable to deploy this contract on Mainnet due to gas fees being high and lack of available capital. Deployment costs were estimated to cost $600 at the time of testing.

However, as this method was fully functional on the test network indicated by the successful transaction seen in Figure 4.23, this would work on Mainnet assuming more profitable opportunities are found. Taking account of the results of the actual *Flash-Low Sell-High* method on Mainnet, it is likely that this method would also result in a number of failed transactions due to high competition.

However, frequency of liquidating loans would also depend on the market sentiment as considered in Chapter 2.4.3. During a period of volatility, particularly a market downturn, this method would be expected to perform well as many opportunities would appear in a short period of time. This would lower the expected competition as there are more opportunities per trader. However, as these trades are often enormously profitable, they are targeted by the most advanced bots, meaning that competition will still be fierce for the opportunities.

## 4.5 Flash Buy-low Sell-high & Buy-low Sell-high

### Overview

Following the unprofitable results of the previous *Flash-Low Sell-High* testing, the author attempted to devise a strategy that would target lesser-known protocols where there would be less competition for the opportunities. The *Flash-Low Sell-High* contract was altered to develop a '*Flash Buy-Low Sell-High'* contract that could be used on any two exchanges, instead of requiring the Uniswap *flash swap* to be used for the initial "buy" part of the trade.

### Implementation

The author selected two exchanges based on specific criterion, including usage and trade pricing mechanism. The author then used the new *Flash Buy-Low Sell-High* smart contract for trading, and a new script to initiate two transactions by calling the exchanges contracts swap methods sequentially from a NodeJS script. There is the risk of failed transactions when executing two transactions independently. It also required some upfront capital; however, the author choose to do this for a specific reason due to how one of the exchanges in question priced swaps. The exchange used volatility to price the trades which appeared to lag behind other exchanges using the standard constant product model. By submitting the transactions separately this volatility could be capitalised on. After initial capital had been acquired using the *Flash-Low Sell-High* contract for executing the trades, the author switched to executing the two trades as separate transactions from a script to capitalise on the volatility pricing mechanism.

The bulk of the code for this section will not be disclosed nor discussed in detail as the author plans to continue to use the method, however the key point to note is that it simply implements a slightly altered version of the pricing script in the *Flash-Low Sell-High* section, and a slightly altered version of the contract. The methods are generalisable and the strategy is what was changed here, only minor modifications were made to the implementation to adjust the code to support different exchange protocols.

**Smart Contract Structure**

The trade sequence is started similarly to the *Flash Liquidation* method, as the Aave flash loan is used (seen in Figure 4.20). Different parameters are passed which are then decoded within the call-back function seen in Figure 4.2. The bulk of the work is then handled in this call-back function in the smart contract after receiving the flash loan from Aave.

Two exchanges are used, one for the "buy-low" and the other for the "sell-high." Uniswap and Sushiswap are used in the example shown in Figure 4.2, however these two methods could be swapped for any two exchanges and their swap methods. The author changed the swap methods to two lesser-known exchanges for the profitable implementation, however the structure and sequence of operations within the contract is identical.

```
function executeOperation(
        address[] calldata assets,
        uint256[] calldata amounts,
        uint256[] calldata premiums,
        address initiator,
        bytes calldata params
    ) external override returns (bool) {
        (
            address addressFlashed,
            address addressWillTradeTo,
            uint256 amountFlashed
        ) = abi.decode(params, (address, address, uint256));

        // approve uniswap contract
        IERC20(addressFlashed).approve(address(uniswapV2Router), amountFlashed);

        // define direction of trade
        address[] memory path = new address[](2);
        path[0] = addressFlashed;
        path[1] = addressWillTradeTo;

        // Swaps to required token to other token
        try
            uniswapV2Router.swapExactTokensForTokens(
                amountFlashed,
                0, // can set this to 0 as will just revert at end if not
profitable and were trying to get MAX out
                path,
                address(this),
                (block.timestamp + 600)
            )
        {} catch Error(string memory reason) {
            emit ErrorHandled(reason);
        } catch {}

        // approve sushiswap contract for trade back
        IERC20(addressFlashed).approve(
            address(sushiRouter),
            IERC20(addressWillTradeTo).balanceOf(address(this))
        );
        // reverse path
        path[1] = addressFlashed;
        path[0] = addressWillTradeTo;

        // Swaps opposite direction
        try
            sushiRouter.swapExactTokensForTokens(
                IERC20(addressWillTradeTo).balanceOf(address(this)),
                0, // can set this to 0 as will just revert at end if not
profitable and were trying to get MAX out
                path,
```

```
                address(this),
                (block.timestamp + 600)
            )
        {} catch Error(string memory reason) {
            emit ErrorHandled(reason);
        } catch {}
        // Calculate Profit
        uint256 profit =
            calculateProfits(
                IERC20(assets[0]).balanceOf(address(this)),
                amounts[0],
                premiums[0]
            );

        require(profit > 0, "!Profitable");
        IERC20(assets[0]).transfer(owner, profit);

        // Approve the Aave contract for repayment
        uint256 amountOwed = amounts[0].add(premiums[0]);
        IERC20(assets[0]).approve(address(_lendingPool), amountOwed);

        return true;
    }
```

*Figure 4.24: Flash Buy-low Sell-high Function Handling Two Swaps before Repaying Loan*

The contract assumes after the swaps the final token held is the token needed to reimburse the loan. However, the *swapToBorrowedToken* function seen in Figure 4.22 could also be added from the *Flash Liquidation* contract to allow for a more generalisable swap. The additional fees on swaps however must be taken into account within the trading script pricing as each Uniswap swap adds an additional 0.3% fee. The two exchanges used for the actual “buy-low” and “sell-high” could then be any two exchanges, therefore their fees and pricing mechanisms would depend on the exchanges implemented.

**Results**

The author implemented both a contract and a script which called transaction methods directly from the bot's wallet. The author selected the exchange carefully and as this method is still viable exact details are omitted in Figure 4.25 below which shows one of the transactions. The key points the author wants to convey is that the methods explored in this paper generalise across other exchanges and that beating front runners is possible, if exchanges are selected with specific criteria.

The profit attained over the period was approximately $20,000 after gas fees were deducted.

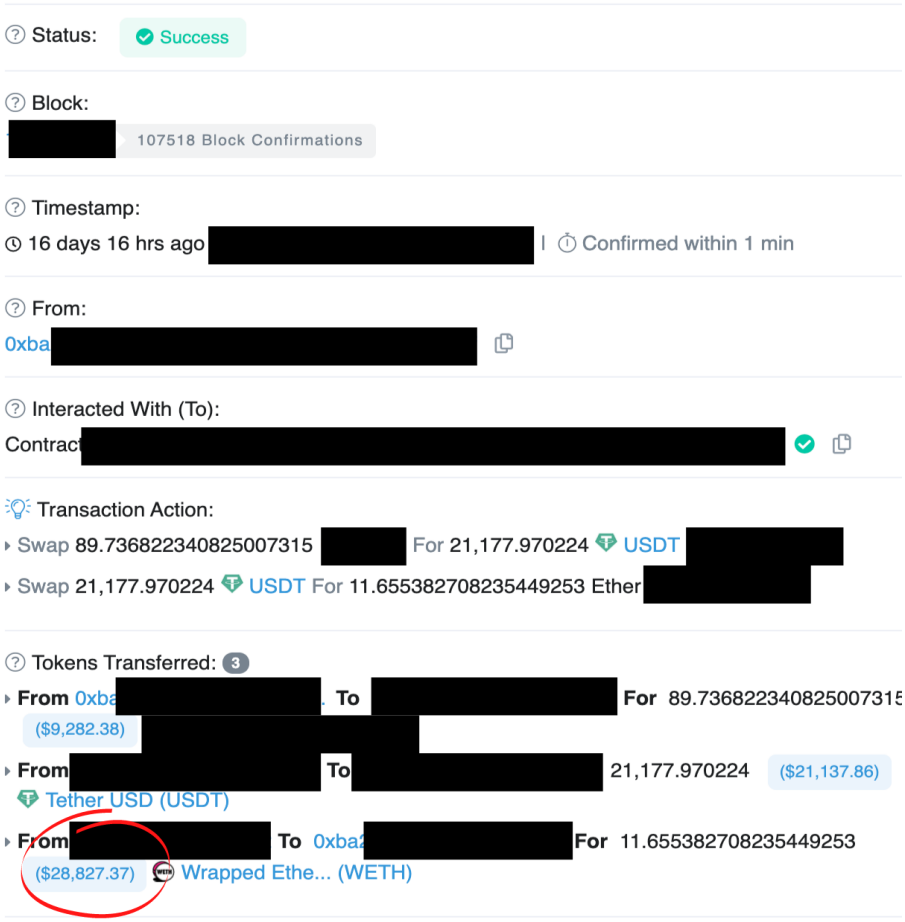


*Figure 4.25: Trading Method Applied to Different Exchange with Total Profit Shown.*

Unfortunately, this is when the authors luck ran out. On a later trade, sometime after Figure 4.25, the bot's wallet ended up finishing a trade on a particularly volatile token. This token subsequently fell 90% in value eliminating almost all of the profits. An optimal strategy would consider reducing risks by not holding volatile tokens.

## 4.6 Triangular Arbitrage

Triangular arbitrage, as discussed in Chapter 2.6.4, was investigated in a decentralised finance context. Concepts were implemented with inspiration from Martin's (2020) blog on centralised cryptocurrency exchange arbitrage. The author used an API that aggregates different decentralised finance exchanges and called it for each combination of pairs of tokens being examined. Following this a weighted directional graph was constructed with the nodes representing currencies and the weights between them the exchange rate. An altered version of the bellman-ford algorithm was used to find any triangular arbitrage opportunities. Ultimately due to time constraints this method was not finished however it also has potential problems and should be approached in a different way for future research that the author will discuss.

**Trading Bot (Incomplete)**

The author adapted the method used by Martin (2019) to run with JavaScript and for decentralised exchanges instead of centralised exchanges. First a URL is generated for each pair of tokens which is used to call an API that aggregates prices across numerous decentralised exchanges to get the best price for swaps between the tokens. Following this a weighted directional graph is created using the negative logarithm of each of the best spot rates as the weight between nodes. This is adapted from Martin's (2019) python implementation.

```
function createDirectedGraphFromPrices(priceObjects,symbols){
    let bestPrices = {}
    let bestExchanges = {

    for(row of priceObjects){
        let bestPrice = 0
        let bestExchange = "N/A"
        for (priceObj of row){
            let price = priceObj['price']
            if(price > bestPrice){
                bestPrice = price
                bestExchange = priceObj['dex']
            }
        }
        let base = row[0]['pair']['base']
        let quote = row[0]['pair']['quote']

        if(bestPrices[base]){

            bestPrices[base][quote] = bestPrice
        }else{

            bestPrices[base] = {}
            bestPrices[base][quote] = bestPrice
        }

        if(bestExchanges[base]){
            bestExchanges[base][quote] = bestExchange
        }else{
            bestExchanges[base] = {}
            bestExchanges[base][quote] = bestExchange
        }
    }
    let graph = initGraph(symbols);
    // convert above to graph
    for (let rowToken of symbols){
        for(let colToken of symbols){
            if(rowToken!=colToken){
                let vertex1 = graph.getVertexByKey(rowToken)
                let vertex2 = graph.getVertexByKey(colToken)
                let edge = new
GraphEdge(vertex1,vertex2,(Math.log(bestPrices[rowToken][colToken]) * -1 ))
                graph.addEdge(edge)
            }
        }
    }
    return graph;
}
```

*Figure 4.26: Converting Prices to Weighted Directional Graph with Negative Logarithmic Weights*

The reasoning behind the negative logarithm being used is best demonstrated with an example When looking for a profitable trading opportunity, a cycle must be found where edge weights all multiply to give a value greater than one, $w1 * w2 * ... * wn > 1$ (Pai, 2019). However, when using graph algorithms such as Bellman Ford, paths are found by summing weights not multiplying weights. The logarithm of the spot rate can therefore be used instead of the actual spot rate as summing the edge logarithmic weights along a path is equivalent to multiplying the rates, and the quantity can be recovered at the end by exponentiating the sum (Martin, 2019). By taking the logarithm of both sides this gives $\log(w1) + \log(w2) + .. + \log(wn) > 0$ (Pai, 2019). The Bellman-Ford algorithm also attempts to find the *minimum* weight paths which can be altered to find negative cycles, therefore the final modification to make here is to use negated logarithmic weights (Martin, 2019) which is $(-\log(w1)) + (-\log(w2)) + .. + (-\log(wn)) < 0$ (Pai, 2019).

To find negative cycles the Bellman-Ford algorithm is then slightly altered. After shortest paths have been identified in the standard implementation, if there is an edge that can be relaxed further, and decrease the value of the shortest path further, then the edge in question is on a negative weight cycle. To find this cycle the path is then walked back along until a cycle is detected and the cyclic portion is returned (Martin, 2019). The JavaScript code for this can be seen below and is adapted from Martin's (2019) python implementation.

```
function bellmanFordAltered(graph, startVertex) {
    let d = new DefaultDict(10000000)
    let p = new DefaultDict(-1)
    let n = graph.getAllVertices().length;
    d[startVertex] = 0

    const allVertices = graph.getAllVertices()
    for(let i=0; i < n-1; i++){
      for(let u of allVertices){
        for(let v of allVertices){
          if(u!=v){
            let edge = graph.findEdge(u,v)
            let weight = edge.weight
            if(d[u] + weight < d[v]){
              d[v] = d[u] + weight
              p[v] = u // updates predecessor
            }
          }
        }
      }
      let allCycles = []
      let seen = new DefaultDict(false)
      for(let u of allVertices){ // find cycles if they exist
        for(let v of allVertices){
          if(seen[v]){
            continue
          }// if we can relax further there must be a neg-weight cycle
          let edge = graph.findEdge(u,v)
          let weight = edge.weight
          if(d[u] + weight < d[v]){
            let cycle = []
            let x = v
            while(true){
              // walk back along preds until a cycle is found
              seen[x] = true
              cycle.push(x)
              x = p[x]
              if(x == v || cycle.includes(x)){
                break
              }
            }
            let index = cycle.indexOf(x) // slice to get cyclic portion
            cycle.push(x)
            allCycles.push(cycle.slice(index).reverse())
          }
        }
      }
      return allCycles
    }
}
```

*Figure 4.27: Modified Bellman-Ford to Find Negative Cycles Adapted from Martin (2021)*

After the optimal path is found for an arbitrage opportunity, a problem occurs. The path is only found for the spot rate between the tokens, but as described previously many of the exchanges implement trading through models such as the constant product model where trade sizing has an impact on the price. This means this method will not work as Bellman-Ford only gives an exact answer for current spot price and as trade size affects price it cannot be solved. There are also numerous types of exchanges with different pricing mechanisms other than the constant product model that would need to be accounted for. A method would need to be devised to optimise trade size accounting for different pricing mechanisms while also finding a profitable arbitrage path.

The author would like to present findings here and a possible solution for future research. Firstly, when looking exclusively at exchanges implementing the constant product model a similar method to that used in the *Flash-Low Sell-High* method to find the trade size could be used. After finding a negative cycle which corresponds to a profitable trading opportunity, optimising the trade size could be solved similarly to Figure 4.4, extending this method to account for more exchanges. For example, taking three exchange pools and three token pairs, one additional step must be added. In the example below these tokens are referred to as $x$, $y$ and $z$. The three starting formulas of the exchanges would be $x_1 * y_1 = k_1$, $y_2 * z_2 = k_2$ and $x_3 * z_3 = k_3$. This could then be rearranged and solved similarly to before.

Pseudo Code for additional Exchange

bestDifference = 0

optimalSize = 0

$\Delta x_1 = 1$

while $\Delta x_1 < x_1$ AND $\Delta x_1 < x_2$

$$\Delta y_1 = \left| \frac{k_1}{x_1 + \Delta x_1} - y_1 \right|$$

$$\Delta z_2 = \left| \frac{k_2}{y_2 + \Delta y_1} - z_2 \right|$$

$$\Delta x_3 = \left| \frac{k_3}{z_3 + \Delta z_2} - x_3 \right|$$

currentDifference = $\Delta x_3 - \Delta x_1$

if currentDifference > bestDifference

bestDifference = currentDifference

optimalSize = $\Delta x_1$

$\Delta x_1 = \Delta x_1 + 1$

This provides a solution to optimise a path once found if all exchanges are governed by constant product formula exchanges. The method could also be extended further for solving longer paths, for example four token pairs. Each of the rearranged constant product formula models for finding the $\Delta$ output values could also be substituted for any other mathematical model that may govern exchanges.

# Chapter 5 | Evaluation

The evaluation section will provide a comprehensive analysis of the results of the methods and discuss challenges and problems uncovered. The development and testing process is also described and a comparison of the current state of the system against the requirements is made. This section concludes with learnings and recommendations for a profitable trading strategy.

## 5.1 Early Challenges

**Testnet Supports**

The author initially planned to test the methods on test networks such as Kovan. These allow users to obtain test versions of Ether and other cryptoassets at no cost. This became problematic for some methods as different protocols use different versions of test tokens, so they are not interchangeable. The author attempted to overcome this problem for the *Flash-Low Sell-High* method by creating his own liquidity pools using the test tokens from one protocol on another, for example using test tokens supported on Uniswap to create a liquidity pool on Sushiswap. The author presumed this would allow the testing of this proof-of-concept method on a test network. However, Sushiswap was not deployed on any test networks so this was impossible. The author attempted to work around this by making the Uniswap contract trade with itself for testing purposes. However, this was impossible as Uniswap prevents against that to protect against a form of attack known as a "*re-entrancy attack*." Ultimately after extensive examination, the author decided to deploy the contracts to the Ethereum Mainnet to test this method in production. Deployment costs were $500 per contract and transaction costs varied from $30 - $120.

### Extortionate Gas Fee Increases

When the author began research for this work in September 2020, the average gas cost for sending a simple transaction was only around $2 and the cost to deploy a relatively large contract, such as the trading contracts which were implemented in this work, would have been approximately $20. The author initially planned to deploy all of the contracts on the Ethereum Mainnet. However, due to the exponential increase in usage, caused primarily by the popularity of decentralised finance, these costs have risen more than 1500% to date, as shown below in Figure 5.1.

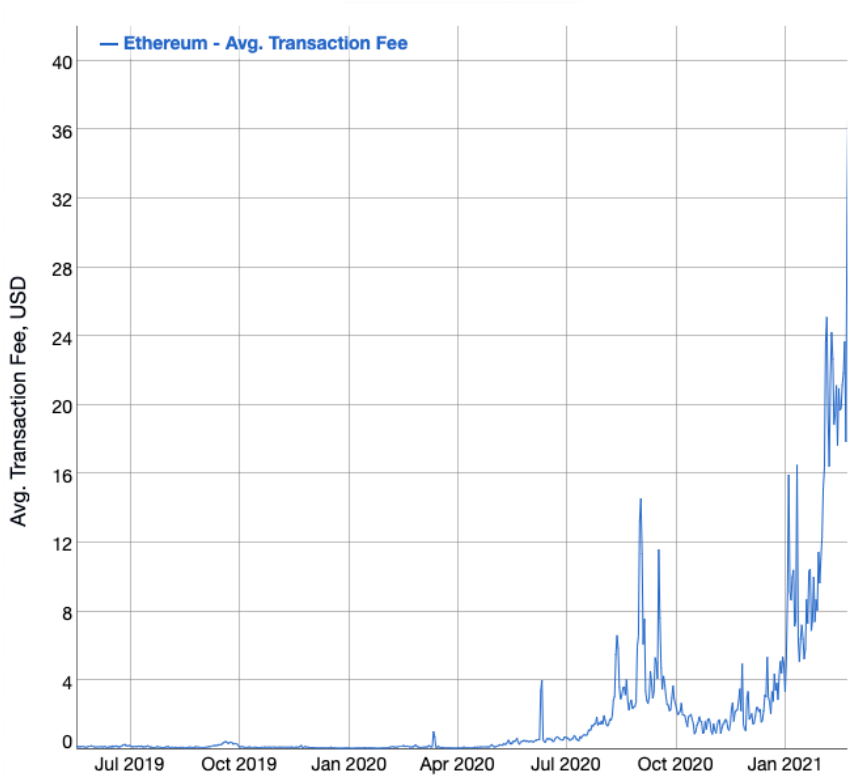


*Figure 5.1: Ethereum Average Transaction Fee over Time (BitInfoCharts, 2021).*

As a result of this, deploying the contracts to Mainnet became extremely expensive. The author felt testing the smart contracts in a simulated environment did not accurately reflect real trading as there would be no competition or front running. Due to this, the author decided to deploy the smallest contracts on Mainnet, the *Flash-Low Sell-High* and the *Flash Buy-Low Sell-High* smart contracts. This had a total transaction cost of 0.3 Ether per contract at the time of deployment, which translated to just under $500 per contract. The author felt this was necessary, however the author did not have adequate capital to deploy the other contracts. Nonetheless, evaluating the impacts of front running by testing with these contracts allowed conclusions to be drawn about how competitive the landscape was. This in turn allowed the author to conclude the viability of all of the methods by using the competition observed when evaluating the methods which were only functional on test networks. Testing in production was also limited by gas fees as failed transaction fees quickly add up.

## 5.2 Fulfilment of Requirements

### 5.2.1 Functional Requirements

The author successfully completed the functional requirements listed, with the primary methods functional on test networks or Mainnet, implementing novel trading methods using new financial instruments.

1. Novel methods successfully implemented
    a. *Flash-Low Sell-High* trading method
    b. *Flash-Liquidation* trading method
    c. *Flash Buy-Low Sell-High* trading method
2. New financial instruments successfully implemented
    a. *Flash Loans* utilised
3. Other Requirements Successfully Implemented
    a. Pricing optimisation
    b. Scripts to evaluate performance

### 5.2.2 Non-Functional Requirements

**Reliability**

The trading scripts to gather data, find opportunities and execute ran reliably and all errors that occurred with unsuccessful transactions were handled and logged to an output file. There is room for improvement with the reliability of data, such as the getting the block late problem discussed previously.

**Performance**

Asynchronous requests could be run in parallel to improve the performance of the proof-of-concept system. In a future implementation it would also be recommended that the language used be changed to something lower level than JavaScript such as C. Smart contracts on Ethereum can also be optimised for gas using lower-level languages such as Yul+ (Fuel Labs, 2021) which would lower gas used thus enabling the usage of higher gas price per unit, ultimately leading to winning the profitable opportunity more often.

**Security**

The deployed contracts on both test networks and Mainnet were limited to being accessed by only the owners for most methods, and only by the particular pool when call-backs were used. The author observed no external parties interacting with the deployed contracts indicating a base level of security was achieved by the implementation.

**Flexibility**

A base level of flexibility was proven as the author adapted the *Flash-low Sell-High* contract and script into the *Flash Buy-Low Sell-High* the method that was described which ultimately led to a profit.

## 5.3 Difficulties Encountered

**Learning Curve**

Prior to this work the author had limited experience with development on blockchain networks such as Ethereum and its associated technologies. There is a steep learning curve with blockchain development, in particular with decentralised finance as it also involves understanding the inner workings of different protocols and the effect of network architecture on the implemented trading methods. A large amount of time was invested into learning the architecture of blockchain and gaining an in-depth understanding before implementation could commence. A large portion of time was also spent researching the protocols with extensive reading of documentation to understand how they operate and how to integrate into them, and finally to implement the trading strategies.

**Poor Development Tools**

Test network supports were limited across utilised protocols. Debugging the production environments is also difficult and costly for failed transactions. This made development both time consuming and challenging. Documentation was also limited or outdated on many protocols, and as seen by the results of the methods a test environment does not give a true reflection of the potential profitability of the strategies due to front running.

# Chapter 6 | Conclusion and Future Work

## 6.1 Conclusion

In the authors opinion, decentralised finance may be the biggest disruption to finance in history. At the start of this work, it was mentioned decentralised finance reached $25 billion in value locked in protocols by January 2021. At the time of concluding this work this figure is $65 billion (Defipulse.com, 2021). The space is growing exponentially, and innovation occurred at speeds the author struggled to maintain pace with during this work. The interoperability of applications is ultimately what empowers this as displayed by the authors ability to integrate with numerous protocols. Trading strategies that require large sums of capital which are usually limited to wealthy corporations have been democratised and opened to anyone with the technical prowess to implement flash loans. Decentralised finance has the potential to transform finance similarly to how the Internet transformed media, allowing small teams of engineers to build applications which capture billions of dollars in value

This work covered the in-depth architecture of decentralised finance from the base level of blockchain to the numerous emerging protocol types and their technicalities. Research of the current state of *DeFi* and potential future impacts of network changes were also considered. The authors implementation of the PoC scripts, smart contracts and evaluation tools then offer unique insight into the current landscape of trading within decentralised finance.

The outcomes of testing in production affirmed the author's beliefs about the competitiveness of the space but the profits attained by the author were a major achievement and offer a glimpse of hope that the competition can be beat.

## 6.2 Future Work

The future of decentralised finance promises open access to financial services to all, and the ability to seamlessly integrate and build applications, regardless of race, gender, nationality or accreditation. This is a stark contrast to traditional finance; however, the often exclusionary and heavily regulated traditional finance is not this way without reason. Regulation offers users protection and helps prevent against malicious behaviour, and as was described with front running in this work, malicious behaviour is already occurring at the expense of users within *DeFi*. Numerous large-scale protocol failures were also discussed. There is a question to be answered here; should developers be responsible for billions of dollars in capital and suffer no consequences when systems fail? Future work may explore these questions, protocol security and the inevitable attempt at regulation, which is likely to be problematic with the decentralised nature of blockchain technology.

Within the context of trading, future work should focus on optimising profitability. This could be achieved by using lower-level languages to reduce gas usage within the smart contracts and to improve the speed of implemented trading scripts. An optimal system architecture should also be researched such as that discussed in Chapter 3.6, accounting for the discussed trade-offs. Improvements could also be made to consider transactions in the Mempool and their effect on exchange rates after they are confirmed. Finally, strategies should be devised to either avoid detection of front running bots or to make the ethically questionable decision of engaging in similar practices.

# Bibliography


Aave.com. 2021. *Aave – Open Source DeFi Protocol*. [online] Available at: <https://aave.com/> [Accessed 2 March 2021].

Adams, H., 2018. UniSwap. [online] Available at: < https://uniswap.org/whitepaper.pdf >

Ahmad, A., Saad, M., Bassiouni, M. and Mohaisen, A., 2018. Towards Blockchain-Driven, Secure and Transparent Audit Logs. *Proceedings of the 15th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*,.

Aiba, Y., Hatano, N., Takayasu, H., Marumo, K. and Shimizu, T., 2002. Triangular arbitrage as an interaction among foreign exchange rates. *Physica A: Statistical Mechanics and its Applications*, 310(3-4), pp.467-479.

Anonymous, 2021. *Rekt - Return to the dark forest*. [online] Available at: <https://www.rekt.news/return-to-the-dark-forest/> [Accessed 7 February 2021].

Back, A. 1997. Hashcash. [online] Available at: <http://www.cypherspace.org/hashcash/>[Accessed 20 April 2021].

Binance Academy. 2021. *What Is Staking? | Binance Academy*. [online] Available at: <https://academy.binance.com/en/articles/what-is-staking> [Accessed 6 March 2021].

BitInfoCharts. 2021. *Ethereum Avg. Transaction Fee Chart*. [online] Available at: <https://bitinfocharts.com/comparison/ethereum-transactionfees.html> [Accessed 1 February 2021].

Blocknative.com. 2020. *Mastering The Mempool: Your Intro To In-Flight Transactions*. [online] Available at: <https://www.blocknative.com/blog/mempool-intro> [Accessed 2 January 2021].

Bouidani, M. M., 2015. Design and Implementation of a Blockchain Shipping Application, Philadelphia: University of Victoria.

Buterin, V., 2013. *Ethereum Whitepaper | ethereum.org*. [online] ethereum.org. Available at: <https://ethereum.org/en/whitepaper/> [Accessed 14 February 2021].

Buterin, V., 2021. An Incomplete Guide to Rollups. [online] Vitalik.ca. Available at: <https://vitalik.ca/general/2021/01/05/rollup.html> [Accessed 1 March 2021].

C., Raja., 2020. Understanding an Ethereum Transaction. [online] Etherscan Information Center. Available at: <https://info.etherscan.com/understanding-an-ethereum-transaction/> [Accessed 3 March 2021].

Cachin, C., 2017. *Blockchain, Cryptography, And Consensus Presentation.* [online] Available at: <https://crypto.unibe.ch/talks/20170622-blockchain-ice.pdf>

Chase, C., 2020. *Twitter*. [online] Twitter.com. Available at: <https://twitter.com/Chris2pherChase/status/1330206780127850497> [Accessed 10 December 2020].

Cordell, R. and Richards, S., n.d. *Proof-of-work (PoW) | ethereum.org*. [online] ethereum.org. Available at: <https://ethereum.org/en/developers/docs/consensus-mechanisms/pow/#top> [Accessed 1 February 2021].

Cordell, R., Maiboroda, A., Richards, S., Cursi, J. and Hotchkiss, G., n.d. *Ethereum Virtual Machine (EVM) | ethereum.org*. [online] ethereum.org. Available at: <https://ethereum.org/en/developers/docs/evm/#state> [Accessed 12 February 2021].

Cordell, R., Richards, S. and Gontijo, A., n.d,. *ERC-20 Token Standard | ethereum.org*. [online] ethereum.org. Available at: <https://ethereum.org/en/developers/docs/standards/tokens/erc-20/> [Accessed 7 February 2021].

Crane, C., 2021. What Is a Hash Function in Cryptography? A Beginner's Guide - Hashed Out by The SSL Store™. [online] Hashed Out by The SSL Store™. Available at: <https://www.thesslstore.com/blog/what-is-a-hash-function-in-cryptography-a-beginners-guide/> [Accessed 2 February 2021].

Criddle, C., 2021. *Bitcoin consumes 'more electricity than Argentina'*. [online] BBC News. Available at: <https://www.bbc.com/news/technology-56012952> [Accessed 13 February 2021].

Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., Breidenbach, L. and Juels, A., 2019. Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges. *arXiv preprint arXiv:1904.05234*.

Daws, R., 2021. Ethereum 2.0 beacon chain launches with over $400m staked. [online] Developer Tech News. Available at: <https://developer-tech.com/news/2020/dec/01/ethereum-2-beacon-chain-launches-400m-staked/> [Accessed 5 April 2021].

Defipulse.com. 2021. DeFi Pulse | The DeFi Leaderboard | Stats, Charts and Guides. [online] Available at: <https://defipulse.com/> [Accessed 29 April 2021].

Deneuville, M., 2021. An in-depth guide into how the mempool works. [online] Medium. Available at: <https://blog.kaiko.com/an-in-depth-guide-into-how-the-mempool-works-c758b781c608> [Accessed 26 April 2021].

Docs.soliditylang.org. 2021. Solidity — Solidity 0.7.4 documentation. [online] Available at: <https://docs.soliditylang.org/en/v0.7.4/> [Accessed 14 February 2021].

Docs.aave.com. 2021. *Liquidations*. [online] Available at: <https://docs.aave.com/developers/guides/liquidations> [Accessed 1 March 2021].

Docs.dfuse.io. 2021. Transaction Lifecycle | dfuse docs. [online] Available at: <https://docs.dfuse.io/ethereum/public-apis/reference/trx_lifecycle/> [Accessed 5 February 2021].

Dothan, M., 2008. Efficiency and arbitrage in financial markets. *International Research Journal of Finance and Economics*, *19*(19), pp.102-06.

Dwork, C., Naor, n. 1993. "Pricing via Processing, Or, Combatting Junk Mail, Advances in Cryptology". CRYPTO'92: Lecture Notes in Computer Science No. 740. Springer: 139–147.

En.wikipedia.org. 2021. *Cryptocurrency*. [online] Available at: <https://en.wikipedia.org/wiki/Cryptocurrency> [Accessed 5 February 2021].

En.wikipedia.org. 2021. *Decentralized application*. [online] Available at: <https://en.wikipedia.org/wiki/Decentralized_application> [Accessed 8 February 2021].

En.wikipedia.org. n.d. *Merkle tree*. [online] Available at: <https://en.wikipedia.org/wiki/Merkle_tree> [Accessed 14 February 2021].

ethereum.org. n.d. *Ethereum Glossary | ethereum.org*. [online] Available at: <https://ethereum.org/en/glossary/#keccak-256> [Accessed 5 February 2021].

ETH Gas Station. 2019. *What is Gwei? - ETH Gas Station*. [online] Available at: <https://ethgasstation.info/blog/gwei/> [Accessed 6 January 2021].

Fama, E., 1970. Efficient Capital Markets: A Review of Theory and Empirical Work, Journal of Finance, 25, 383-417.

Frankenfield, J., 2018. *Distributed Ledger Technology*. [online] Investopedia. Available at: <https://www.investopedia.com/terms/d/distributed-ledger-technology-dlt.asp> [Accessed 6 December 2020].

Frankenfield, J. and Anderson, S., 2021. Proof of Work (PoW). [online] Investopedia. Available at: <https://www.investopedia.com/terms/p/proof-work.asp> [Accessed 21 April 2021].

Frankenfield, J. and Anderson, S., 2021. Ethereum. [online] Investopedia. Available at: <https://www.investopedia.com/terms/e/ethereum.asp> [Accessed 6 April 2021].

Foxley, W., 2021. *Ethereum's 'EIP 1559' Fee Market Overhaul Greenlit for July*. [online] Nasdaq.com. Available at: <https://www.nasdaq.com/articles/ethereums-eip-1559-fee-market-overhaul-greenlit-for-july-2021-03-05> [Accessed 29 March 2021].

Foxley, W., 2021. *Ethermine Adds Front-Running Software to Help Miners Offset EIP 1559 Revenue Losses - CoinDesk*. [online] CoinDesk. Available at: <https://www.coindesk.com/ethermine-adds-front-running-software-to-help-miners-offset-eip-1559-revenue-losses> [Accessed 24 March 2021].

Fuel Labs, 2021. Introducing: Yul+—A new low-level language for Ethereum. [online] Medium. Available at: <https://fuellabs.medium.com/introducing-yul-a-new-low-level-language-for-ethereum-aa64ce89512f> [Accessed 2 April 2021].

Ganti, A., 2021. Cash-And-Carry Trade Definition. [online] Investopedia. Available at: <https://www.investopedia.com/terms/c/cashandcarry.asp#:~:text=Key%20Takeaways-,A%20cash%2Dand%2Dcarry%20trade%20is%20an%20arbitrage%20strategy%20that,a%20futures%20or%20options%20contract.> [Accessed 3 March 2021].

Gasnow.org. 2021. Gas Now — ETH Gas Price quotation system based on Pending transactions. [online] Available at: <https://www.gasnow.org/> [Accessed 3 February 2021].

Giesen, M., 2020. *Learning Ethereum: What is a Hash Function and why is it important?*. [online] Unvetica | Full Stack Development & Marketing Agency. Available at: <https://unvetica.com/learning-ethereum-hash-function/> [Accessed 8 February 2021].

Gudgeon, L., Perez, D., Harz, D,. Livshits, B., Gervais, A. 2020. The decentralized financial crisis. In 2020 Crypto Valley Conference on Blockchain Technology (CVCBT), pages 1–15. IEEE,

Han, E., 2018. Intra-exchange Cryptocurrency Arbitrage Bot. [online] Available at: <https://scholarworks.sjsu.edu/etd_projects/655/> [Accessed 3 March 2021].

Harari, Y., 2015. *Sapiens: A brief History of Humankind*. Harper Collins USA, Chapter 10.

Hertzog, E., Benartzi, G., 2017. *Bancor protocol - continuous liquidity and asynchronous price discovery for tokens through their smart contracts; aka smart tokens*. Available at: <https://whitepaper.io/document/52/bancor- whitepaper>

Hewa, T., Hu, Yining., Liyanage, M., Kanhare, S., Ylianttila, M., 2021. Survey on Blockchain based Smart Contracts: Technical Aspects and Future Research. IEEE Access. 10.1109/ACCESS.2021.3068178.

Info.uniswap.org. 2021. Uniswap Info. [online] Available at: <https://info.uniswap.org/home> [Accessed 29 April 2021].

Investinblockchain.com. 2017. *How Does Cryptography Protect The Blockchain?*. [online] Available at: <https://www.investinblockchain.com/how-does-cryptography-protect-blockchain/> [Accessed 7 January 2021].

Investopedia. 2021. *Stablecoin*. [online] Available at: <https://www.investopedia.com/terms/s/stablecoin.asp> [Accessed 5 February 2021].

Investopedia. 2021. *What Is High-Frequency Trading?*. [online] Available at: <https://www.investopedia.com/ask/answers/09/high-frequency-trading.asp> [Accessed 29 January 2021].

Investopedia. 2021. *Order Book*. [online] Available at: <https://www.investopedia.com/terms/o/order-book.asp> [Accessed 9 February 2021].

Investopedia. 2021. *What Is Front-Running in Stocks?*. [online] Available at: <https://www.investopedia.com/terms/f/frontrunning.asp> [Accessed 10 February 2021].

Kisagun, C., 2019. Preventing DEX Front-running with Enigma. [online] Medium. Available at: <https://blog.enigma.co/preventing-dex-front-running-with-enigma-df3f0b5b9e78> [Accessed 3 March 2021].

Lee, D., 2015. *Handbook Of Digital Currency*. Amsterdam [etc.]: Academic Press.

Lewis, M., 2014. *Flash boys*. New York: W.W. Norton.

Liu, C., 2008. The Dark Forest.

Luu, L., and Velner, Y., 2017. "KyberNetwork: A trustless decentralized exchange and payment service." [online] Available at: <https://whitepaper.io/document/43/kyber-network-whitepaper> [Accessed 9 February 2021].

Madeira, A., 2020. *Ethereum 'Flippens' Bitcoin To Become The Most Used Blockchain*. [online] Cointelegraph. Available at: <https://cointelegraph.com/news/ethereum-flippens-stablecoins-to-become-the-most-used-blockchain> [Accessed 28 November 2020].

Mahajan, P., Jadhav, V. and Salian, Y., 2020. Currency Arbitrage Detection. *International Journal of Innovative Science and Research Technology*, 5.

Manuskin, A., 2020. *Ethology: A Safari Tour in Ethereum's Dark Forest*. [online] ZenGo - Bitcoin & Cryptocurrency Wallet. Available at: <https://www.zengo.com/ethology-a-safari-tour-in-ethereums-dark-forest> [Accessed 28 January 2021].

Martin, R., 2019. Graph algorithms and currency arbitrage, part 2 · Reasonable Deviations. [online] Reasonabledeviations.com. Available at: <https://reasonabledeviations.com/2019/04/21/currency-arbitrage-graphs-2/> [Accessed 4 March 2021].

Meegan, X. and Koens, T., 2021. Lessons Learned from Decentralised Finance (DeFi).

Merkle, R., 1988. A Digital Signature Based on a Conventional Encryption Function. Advances in Cryptology — CRYPTO '87, pp.369-378.

Millman, R., 2021. What is Ethereum 2.0 and Why Does It Matter? - Decrypt. [online] Decrypt. Available at: <https://decrypt.co/resources/what-is-ethereum-2-0> [Accessed 6 April 2021].

Moen, K., Richards, S. and Gontijo, A., n.d,. *ERC-721 Non-Fungible Token Standard | ethereum.org*. [online] ethereum.org. Available at: <https://ethereum.org/en/developers/docs/standards/tokens/erc-721/> [Accessed 9 February 2021].

Muzzy, E., 2020. *What Is Proof Of Stake? | Consensys*. [online] ConsenSys. Available at: <https://consensys.net/blog/blockchain-explained/what-is-proof-of-stake/> [Accessed 2 December 2020].

Nakamoto, S., 2008. *Bitcoin: A Peer-To-Peer Electronic Cash System*. [online] Bitcoin.org. Available at: <https://bitcoin.org/bitcoin.pdf> [Accessed 15 November 2020].

Nie, Z., Cordell, R., Richards, S. and Hotchkiss, G., n.d. *Intro to Ethereum | ethereum.org*. [online] ethereum.org. Available at: <https://ethereum.org/en/developers/docs/intro-to-ethereum/> [Accessed 11 February 2021].

Obadia, A., 2020. Flashbots: Frontrunning the MEV Crisis. [online] Medium. Available at: <https://medium.com/flashbots/frontrunning-the-mev-crisis-40629a613752> [Accessed 2 March 2021].

Oved, M., Mosites, D., 2017. *"Swap: A Peer-to-Peer Protocol for Trading Ethereum Tokens."* Available at: <https://swap.tech/pdfs/SwapWhitepaper.pdf>

Pai, A., 2019. Currency Arbitrage using Bellman Ford Algorithm. [online] Medium. Available at: <https://anilpai.medium.com/currency-arbitrage-using-bellman-ford-algorithm-8938dcea56ea> [Accessed 27 April 2021].

Page, C., Finlay, D., Eldar, G., Mosites, D. and Alonso, M., 2020. *Defi By Consensys Webinar*. [online] Available at: <https://www.youtube.com/watch?v=0CIdJj5Ngx4> [Attended 9 December, 2020].

Peaster, W., 2021. From Ethereum to Binance Smart Chain. [online] Defiprime.com. Available at: <https://defiprime.com/binance-smart-chain> [Accessed 6 April 2021].

Perez, D., Werner, S.M., Xu, J. and Livshits, B., 2020. Liquidations: DeFi on a Knife-edge. arXiv preprint arXiv:2009.13235.

PoolWatch. 2021. Best Ethereum Mining Pools for 2021 - PoolWatch.io. [online] Available at: <https://www.poolwatch.io/coin/ethereum> [Accessed 2 April 2021].

Popescu, AD., 2020. "Decentralized Finance (Defi) – The Lego Of Finance," Social Sciences and Education Research Review, Department of Communication, Journalism and Education Sciences, University of Craiova, vol. 7(1), pages 321-348, July.

Powers, B., 2020. *New Research Sheds Light on the Front-Running Bots in Ethereum's Dark Forest - CoinDesk*. [online] CoinDesk. Available at: <https://www.coindesk.com/new-research-sheds-light-front-running-bots-ethereum-dark-forest> [Accessed 26 January 2021].

Qin, K., Zhou, L. and Gervais, A., 2021. Quantifying Blockchain Extractable Value: How dark is the forest?. *arXiv preprint arXiv:2101.05511*.

Quiknode.io. 2021. Web3 Endpoint Benchmarking Tool - QuikNode. [online] Available at: <https://www.quiknode.io/compare> [Accessed 1 February 2021].

Rattanaveerachanon, A., Cordell, R., Richards, S., Cursi, J. and Wackerow, P., n.d. *Transactions | ethereum.org*. [online] ethereum.org. Available at: <https://ethereum.org/zh/developers/docs/transactions/> [Accessed 9 February 2021].

Robinson, D., 2020. *Ethereum Is a Dark Forest*. [online] Medium. Available at: <https://medium.com/@danrobinson/ethereum-is-a-dark-forest-ecc5f0505dff> [Accessed 4 March 2021].

Ross, S. and Dybvig, P., 1989. Arbitrage. *Finance*, pp.57-71. Available at: <10.1007/978-1-349-20213-3_4>

Russel, J., 2019. TechCrunch is now a part of Verizon Media. [online] Techcrunch.com. Available at: <https://techcrunch.com/2019/05/10/binance-security-hack/?guccounter=1&guce_referrer=aHR0cHM6Ly9zZWxma2V5Lm9yZy8&guce_referrer_sig=AQAAAD3GrWeP7uSE-e82NxofmUiR8kmgTYiLNnhChP01mXAp6sK9dK4Dtz9BLeqG_7oQNNpZtTiuvArZZJNj-

nppvLoCyd1Wl5TR969CnTUkYG-
0r2VXaXfzial9XswgQJWa84aec9FxAUsrypSNCbDchCmNK0825-7iqCefWlRiTO_e>
[Accessed 20 April 2021].

Schär, F., 2020. Decentralized Finance: On Blockchain- and Smart Contract-based Financial Markets. *SSRN Electronic Journal*,.

Schueffel, P., Groeneweg, N., 2019. Evaluating Crypto Exchanges in the Absence of Governmental Frameworks - A Multiple Criteria Scoring Model.

Swanson, T., 2015. *Permissioned Distributed Ledgers*. https://www.the-blockchain.com/wp-content/uploads/2016/04/Permissioned-distributed-ledgers.pdf.

Szabo, N., 1994. *Smart Contracts*. [online] Fon.hum.uva.nl. Available at: <https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html> [Accessed 21 November 2020].

Tal, Y., 2018. *Introducing The Graph*. [online] Medium. Available at: <https://medium.com/graphprotocol/introducing-the-graph-4a281b28203e> [Accessed 4 March 2021].

Tesař, E., Cordell, R., Nolan, D. and Richards, S., n.d. *Ethereum accounts | ethereum.org*. [online] ethereum.org. Available at: <https://ethereum.org/en/developers/docs/accounts/> [Accessed 5 February 2021].

Thealgorists.com. 2021. *Arbitrage using Bellman-Ford Algorithm - TheAlgorists*. [online] Available at: <https://www.thealgorists.com/Algo/ShortestPaths/Arbitrage> [Accessed 15 January 2021].

The Graph. 2020. *The Graph*. [online] Available at: <https://thegraph.com/> [Accessed 3 March 2021].

Truffle Suite. 2021. Truffle | Overview | Documentation | Truffle Suite. [online] Available at: <https://www.trufflesuite.com/docs/truffle/overview> [Accessed 3 March 2021].

Uniswap.org. 2021. *Uniswap | How Uniswap works*. [online] Available at: <https://uniswap.org/docs/v2/protocol-overview/how-uniswap-works> [Accessed 5 January 2021].

Vogelsteller, F., Buterin, V., "EIP-20: ERC-20 Token Standard," *Ethereum Improvement Proposals*, no. 20, November 2015. [Online serial]. Available: https://eips.ethereum.org/EIPS/eip-20.

Wackerow, P., Richards, S. and Cordell, R., n.d. *Consensus mechanisms | ethereum.org*. [online] ethereum.org. Available at: <https://ethereum.org/en/developers/docs/consensus-mechanisms/#top> [Accessed 1 February 2021].

Wackerow, P., Cordell, R., Stockinger, A. and Richards, S., n.d. *Proof-of-stake (PoS) | ethereum.org*. [online] ethereum.org. Available at: <https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/#top> [Accessed 1 February 2021].

Wang, D., Wu, S., Lin, Z., Wu, L., Yuan, X., Zhou, Y., Wang, H. and Ren, K., 2020. Towards understanding flash loan and its applications in DeFi ecosystem. *arXiv preprint arXiv:2010.12252*.

Warren, W., Bandeali, A., 2018. 0x: “An open protocol for decentralized exchange on the Ethereum blockchain.” Available at < https://0x.org/pdfs/0x_white_paper.pdf>

Werner, S.M., Gudgeon, L., Perez, and W. J. Knottenbelt., 2020. “Defi protocols for loanable funds: Interest rates, liquidity and market efficiency,” in Proceedings of the 2nd ACM Conference on Advances in Financial Technologies. p. 92–112.

Werner, S.M., Perez, D., Gudgeon, L., Klages-Mundt, A., Harz, D. and Knottenbelt, W.J., 2021. SoK: Decentralized Finance (DeFi). arXiv preprint arXiv:2101.08778.

Worsley, N., 2021. Defi-Cartel/salmonella. [online] GitHub. Available at: <https://github.com/Defi-Cartel/salmonella> [Accessed 9 April 2021].

Young, M., 2020. *What Vitalik Buterin Had to Say on ETH 2.0 Launch Day*. [online] Coingape. Available at: <https://coingape.com/what-vitalik-buterin-had-to-say-on-eth-2-0-launch-day/> [Accessed 9 February 2021].

Zetzsche, D., Arner, D. and Buckley, R., 2020. Decentralized Finance. *Journal of Financial Regulation*, 6(2), pp.172-203.

Zheng, Z., Xie, S., Dai, H., Chen, X. and Wang, H., 2017. An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends. *2017 IEEE International Congress on Big Data (BigData Congress)*.

Zhou, L., 2020. *Demystify the dark forest on Ethereum—Sandwich Attacks*. [online] Medium. Available at: <https://medium.com/coinmonks/demystify-the-dark-forest-on-ethereum-sandwich-attacks-5a3aec9fa33e> [Accessed 1 March 2021].

Zhou, L., Qin, K., Torres, C.F., Le, D.V. and Gervais, A., 2020. High-Frequency Trading on Decentralized On-Chain Exchanges. *arXiv preprint arXiv:2009.14021*.