Research and Implementation of Multiple Blockchain Byzantine Secure Consensus Protocols for Robot Swarms

Agnes Cameron, Mark Payne, Bruno Prela

May 16, 2018

Abstract

Distributed consensus mechanisms are key in ensuring the integrity of decisions made by decentralized systems. These mechanisms must be robust to bad actors, but this robustness should not come at a cost to efficiency and scalability. In blockchain systems, the development of heterogeneous consensus mechanisms might be a means of avoiding the inherent scaling problems in the proof of work algorithm. Methods by which these consensus mechanisms might interface with one another – forming a so-called 'multi-chain' system – are currently of great interest. We present a proof-of-concept for a multi-chain system in the area of swarm robotics.

1 Introduction

Recent advances in blockchain technology have led to an expansion beyond cryptocurrencies, namely to applications that require forms of distributed consensus. At the same time, there has been growing interest in the field of swarm robotics for applications in farming[1], search-and-rescue[2], and delivery systems. For practical reasons these systems often need to operate in a decentralized or distributed manner, and thus they require protocols that are Byzantine fault tolerant (named after the famous Byzantine Generals Problem [3]).

We build on the work of Strobel et al.[4] to investigate and implement a multi-blockchain Byzantine fault-tolerant consensus protocol for a swarm robotics application.

1.1 Blockchains as Distributed Consensus Mechanisms

In 2008, the publication of Satoshi Nakamoto's Bitcoin: a Peer-to-Peer Electronic Cash System [5] created the first practical implementation of a blockchain system. The blockchain acts as a distributed ledger for transactions between peers in the network, with every node in that network carrying a full copy of this ledger.

A powerful concept presented by blockchain systems is the idea of distributed consensus, where agreement can be reached among a number of decentralized peers. Bitcoin uses a consensus mechanism called proof of work (PoW), which uses computational expense to prevent bad actors from changing the state of the system through illegal means.

In 2015, Vitalik Buterin released Ethereum [6]. Ethereum differs from Bitcoin in that it allows for the execution of Turing complete code (written in a language called Solidity) as part of the blockchain. This code is defined by a set of entities known as 'Smart Contracts'. The concept of smart contracts dates back to Nick Szabo in 1994, who envisioned a means of allowing complex social forms of agreement to be encoded in a distributed digital system [7]. Ethereum also currently uses the PoW consensus algorithm employed by Bitcoin, though Buterin has declared that the currency will soon adopt a hybrid Proof-of-Stake (PoS) and PoW model [8].

1.2 Scalability Problem with Blockchains

Inherent in PoW are issues with scaling and speed. By design, PoW algorithms are slow, as the system must wait for each node in the network to come to consensus, on top of the time required for mining. For example, in Bitcoin the difficulty of mining a block is purposely set so that on average a block is mined every 10 minutes, allowing enough time for the block to be communicated across the network. Even through condensing multiple transactions per block, this difficulty setting results in a limit on transactions of around 3/sec for Bitcoin (compared to Visa's 45000/sec).

Numerous proposals (PoS included) have been made to rectify this. For example, in PoS blocks are appended to the blockchain by a small set of predetermined validators who have some sort of stake in the transaction [9]. PoS systems are considerably more environmentally friendly, as they don't require the computing power of PoW–they simply rely on the agreement of the nodes. However, the state change of the system is still tied to the consensus protocol, which means that an ultimate limit is placed by the time it takes to agree between the nodes.

Another issue is the size of the chain that must be stored. Currently in both PoW and PoS systems, each full node stores the entire state of the blockchain, to verify the validity of added blocks. In Ethereum, this is at least 20GB (though this changes depending on the sync mode used, and can be up to 385GB) while Bitcoin's blockchain was around 150 GB in size by the end of 2017[10]. This storage requirement can be overwhelming for smaller systems, and poses a problem for expanding blockchain technology into IoT devices and swarm robotics systems.

1.3 Proposed solutions to the Scalability Problem

Inherent in both PoW and PoS systems is an absolute limit imposed by the fact that the state of the system is tied to the consensus mechanism. Thus, the demand made by the network for each state transition can never be higher than the demand made of a single node. There are several major solutions to this scalability problem that have been proposed in the context of Ethereum, including sharding, the Raiden network and Plasma chains.

Sharding divides the nodes on the Ethereum blockchain into smaller sub groups, within which miners compete to verify group transactions, allowing multiple transactions to be validated in parallel across different subgroups. This also diminishes the total size of the blockchain that must be stored at each node[11]. By contrast, the Raiden network proposes a method known as 'state channels' – moving the majority transactions off-chain, allowing them to be processed immediately by a collection of nodes that establish these channels amongst themselves. This has advantages for of IoT and microtransaction -based systems, as it does not involve a wait for block confirmation[12][11].

Plasma proposes a system of hierarchical side-chains, managed by smart contracts. Each ‘child’ chain periodically relays information to its ‘parent’, and in turn that information is relayed up the tree, eventually leading back to the root Ethereum chain[11].

1.4 Multi-Blockchain Protocols

Similar in proposition to Plasma, there have also been proposals for smart-contract mediated systems that go beyond the Ethereum blockchain, allowing heterogeneous consensus protocols to interface with one another. These systems have a pragmatic advantage in that they can provide a link between different forms of governance (e.g. PoW, PoS, FBA), and provide interoperability between public and private blockchain systems[13]. The Cosmos system proposes a hierarchical system not dissimilar to side-chain and Plasma solutions, which uses a master ‘hub chain’ and ‘zoned chains’. Multiple heterogeneous chains operate in ‘zones’, though the incentivization of validators for these chains remains an unsolved problem[14]. Polka Dot proposes a more complex system that leverages shared security between chains, navigating a criticism of multichain systems that security is compromised by reducing the number of agents per chain.

1.5 Swarm Robotics

Forms of decentralized and peer-to-peer agreement are also highly relevant in swarm robotics applications. Swarm robots are particularly capable in solving problems covering a large physical area, requiring peer-to-peer co-ordination to communicate effectively. Increasingly as these systems are implemented in real-world scenarios, we might expect to see examples where swarms must not only coordinate robustly amongst themselves, but with other swarms with different modes of agreement.

1.6 Security Concerns in Swarm Robotics

Whilst many swarm robotic systems cite fault tolerance as a key feature, many systems do not show robustness in the presence of malicious actors. Valentini et. al[15] propose a series of federated agreement protocols for generating consensus among swarms of robots. Although these voting protocols are fault-tolerant, it is demonstrated by[4] that these systems are not robust to the presence of malicious actors.

Strobel et. al[4] implement the consensus mechanisms proposed by Valentini et. al[15] in a Byzantine fault-tolerant blockchain-based system. In this system, each robot votes on what they believe to be the state of the system, corroborated with information from their local peers (as before), by submitting a smart contract. Each robot is run as a separate geth node on the network, and peers are added and dropped depending on the local group. By taking the longest blockchain in any peer group as the most accurate representation of the consensus state, and blacklisting robots submitting old or bad blocks, this system demonstrates significantly higher resistance in the face of bad actors than that proposed in[15].

2 Methods

Initially, we had hoped to use and develop the code from Strobel et. al, as this project was open-sourced after the publication of their paper. However, the codebase includes a large amount of extra complexity that makes it difficult to build and edit. In the interests of effectiveness (wanting to learn about coding multi-blockchain systems, rather than debugging other people's code), we opted to write our own, simpler system from scratch. The Strobel code did, however, provide us a useful framework to start our investigation, and we are grateful to the authors for its publication.

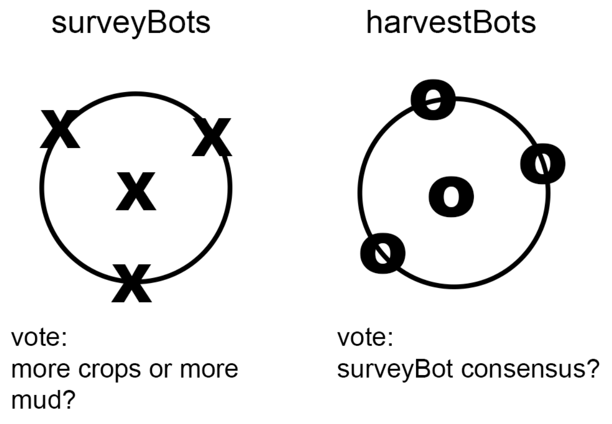
Below we describe our simulation, including the game for our tested swarm robotics application, as well as our codebase to develop a Javascript web application to perform and visualize this simulation.

As a final note, this investigation is intended as a proof-of-concept, rather than something that should be implemented wholesale in a real system.

2.1 Harvesting Game Definition

To implement our proposed multi-blockchain consensus protocol, we simulated a game called the Harvesting Game, which is defined as follows.

Figure 1: *Voting in the Harvest Game*



The game occurs on a square grid meant to represent a large farming field. Each of the squares in the grid can take on one of the two possible attributes: dirt or crops. The field is to be maintained by a two swarms of robots, who collaborate and monitor the percentage of dirt-vs-crops, and decide when to harvest. We will assume that the field is sufficiently large such that the robots can only communicate with their nearest neighbors, and thus this consensus must be reached in a distributed manner.

The first type of robots are surveyBots, which are ultimately responsible for building a model of an environment. They can move relatively quickly, and ultimately must come to a consensus about the majority state of the grid. To come to this consensus, these robots vote on a private blockchain.

The other type are the harvestBots, which are responsible for deciding whether or not to farm the field when they agree that the surveyBots have reached a consensus about the state. To do this, these robots operate on a different private blockchain to the surveyBots, but in order to know when to harvest, they have to communicate robustly with the surveyBots.

The harvestBots make calls to the surveyChain directly, using a contract call, and then report back to their group on what information they gather. This is complicated by the fact that members of both groups might be Byzantine. However, as non-byzantine surveyBots should all report the same values (as they are reporting the value of the last block to be appended to the chain), by using a consensus mechanism among the harvestBots, false values should be weeded out.

In this system, the consensus protocol implemented by each blockchain is not the consensus protocol implemented by the robots. This decoupling is deliberate. Rather, the robots use the blockchain as a means of decentralized voting upon a concept that each agent cannot have absolute knowledge of. Because of this, an honest robot might submit an incorrect vote: it is the collective voting of the robots which allows consensus on the state of the system to be reached.

2.2 Coding Overview

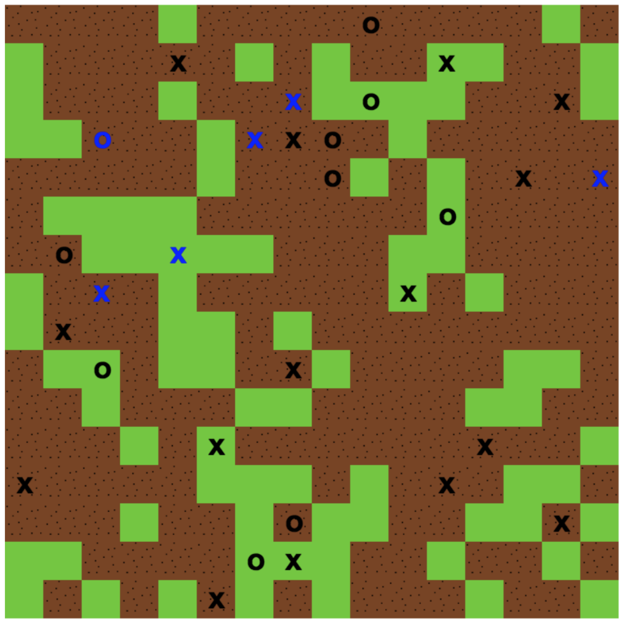
The code for this project can be found at `https://github.com/brunoprela/multiple_blockchain_swarm_robotics` (along with instructions on how to deploy it). The code is essentially a Javascript web application with a front-end for visualizing the simulation while most of the consensus protocols and smart contracts being handled on the back-end. Figure 2 shows the visual layout of the simulation as defined in `client/index.html`. As shown in the figure, the field is represented by a 16x16 grid composed of squares for dirt and crops. Different shapes are used for each type of robot, and different colors are used to distinguish byzantine robots from each other.

In the simulation, the survey robots (with their ability to move quickly and observe the state of the square directly underneath them) first move about the grid for 10 steps (in our case the steps are in random directions). The harvestBot swarm moves once every 10 timesteps. At this point, the harvester robots communicate with the survey bots and try to come to a consensus as to the state of the surveyBot system.

2.3 Smart-Contract Based Distributed Consensus Mechanism in Solidity

To reach a consensus about the state of the field (majority crops or dirt), the swarm robots execute smart contracts (written in solidity) that run on a private Ethereum network. After 10 steps surveying the grid, each robot

Figure 2: *View of the Simulation Environment*



Grid showing honest surveyBots (black 'x'), honest harvestBots (black 'o'), and Byzantine surveyBots and harvestBots (blue 'x' and 'o' respectively). The green grid squares represent healthy crops, the brown represent dirt.

forms an opinion about the state of the field. Instead of submitting their own opinion though, instead each survey robot polls the opinions of the its nearest neighboring robots that it can communicate with and chooses the most popular opinion (including counting its own) as its vote. This is known as majority voting, as described first by Valentini [15] and mentioned later again by Strobel [4].

These votes are then submitted to the smart contract, defined in `smart contracts /surveyVoting.sol`. The smart contract then determines the majority vote submitted by all survey robots, and if all votes are for the majority, returns that a consensus has in fact been reached.

The smart contract also outlines a procedure for identifying and weeding out Byzantine robots. After each round of voting, the smart contract makes notes of each robot address that submitted a vote for the minority opinion, and increments a counter associated with this address of the number of times this has occurred. After enough rounds have passed (arbitrarily set at 4), if the counter for a certain address has exceed a number greater than half the number of rounds, then the robot is deemed Byzantine and is prevented from voting again in future rounds.

The idea behind this scheme is simple–(barring a 51% attack) we expect to be able to ID Byzantine actors by the fact that they continue to vote in the minority, which is especially obvious when surrounded by robots who have the majority opinion. While this scheme is simple, it is potentially vulnerable when there are a significant number of adversarial robots (as opposed to generally faulty robots), though this will be discussed later on.

A similar scheme is used for the harvester robots, implemented in the smart contract `smart contracts/harvestVoting.sol`. The smart contract is similar to the one used by the survey robots, with the main exception being the variable that is being voted on.

2.4 Overview of Backend

The backend of the application consists of the blockchain management tools and the database. The `smart contracts` folder contains the smart contracts used by the system, and instructions to deploy them. We chose to deploy our smart contracts on a test network using the Truffle suite of tools (Ganache-cli specifically) because of the ease with which they allow you to develop a test network and communicate with it through Web3. The obvious drawback to having such simplicity is that we had less control over the nature of our test network (although the detailed config files help). The reason for taking this tradeoff was that the Go implementation of Ethereum (geth, a demonstration of which is shown in Figure 4), which is used for interacting with the main Ethereum chain as well as for interacting with your own private test chains in greater depth, is difficult to work with and requires using much lower level language and much more complex considerations. In fact, it is the primary reason we decided to develop our own simulation environment. We discuss the possible extensions to the backend which could help others get started researching this topic using our code more easily in Section 3.5.

The database section is mostly contained in the `server` folder, and contains code for a Node.JS server with models for handling robots, experiments and swarms, as well as a fully developed GraphQL API. This section is not actually in use, and most of our state management is happening on the clientside. We would have liked to move our logic to a client-server logic structure so that we could keep more in-depth data on our experiments and offload some of the logic from the clientside. As the scale of our experiment was relatively small, we decided against it and invested more time on other aspects of the project. One of the problems we found the Strobel codebase was having was simply the storage of experimental data (which geth makes nearly impossible to do neatly and outside of printing to files and reading them back up again with other code). Discussion of why we put this code in here and how we wish to extend it in the future can be found in Second 3.5.

2.5 Experimental Overview

In order to test the system, the simulation was run with varying numbers of Byzantine robots, and a variable difficulty problem in identifying the dominant ‘floor’ tile.

Overall, we were looking to see if the simulation would correctly identify and weed out the Byzantine robots (easily seen as the robots would turn yellow) without incorrectly identifying any truthful robots (in our code this would cause the robot labels to turn red).

Much of our choices unfortunately were governed by the fact that Ethereum effectively regulates the amount of computation that can be required in a smart contract through the use of ’gas’. For example, each elementary operation in a smart contract has an associated amount of gas required to execute the contract, which must be payed by the parties involved in the contract to the miner (that way the reward for the miner is adjusted by the computational cost).[16]

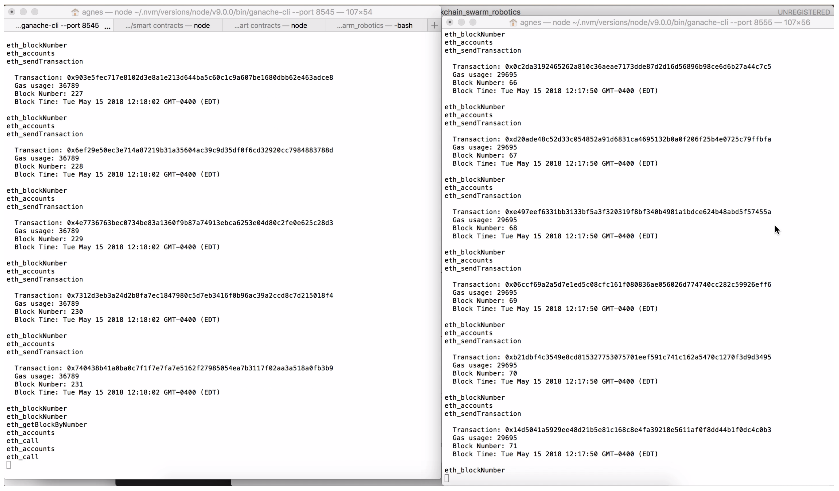
It is very easy for a contract to quickly run out of gas, as was the case for our system. In fact, one source looked at comparing the cost of executing code in a smart contract (based on the current price of gas in Ethereum) and concluded that computation in Ethereum is about $400 million times as expensive as the cost of running computation on commercial servers.[16] Thus, it does not take a whole lot of computation before the cost of the contract becomes too high.

Because of this, we required a low number of rounds (just 4) before weeding out Byzantine robots, and used a threshold of 50% of the rounds in the minority for identifying Byzantine robots. This allowed for the system to reach consensus quickly so as not to exceed the gas limit on the contracts.

3 Discussion

Overall we believe this system satisfied the aim of managing an interaction between two blockchains, while also presenting a valid proof-of-concept of the merit of multi-blockchain technology in the field of swarm robotics. As we review our results, though, we also note that we suffered significant drawbacks which prevented us from developing a truly Byzantine fault tolerant system, and from developing the level of software infrastructure around our experiments that we would have liked to simulate more realistic settings.

Figure 3: *Successful running of two heterogeneous blockchains in ganache-cli*


agnes — node ~/.nvm/versions/node/v9.0.0/bin/ganache-cli --port 8545 — 107×54
...ganache-cli --port 8545
.../smart contracts — node
...art contracts — node
...arm_robotics — -bash
kchain_swarm_robotics
UNREGISTERED
agnes — node ~/.nvm/versions/node/v9.0.0/bin/ganache-cli --port 8555 — 107×56

```
eth_blockNumber
eth_accounts
eth_sendTransaction

  Transaction: 0x903e5fec717e8102d3e8a1e213d644ba5c60c1c9a607be1680dbb62e463adce8
  Gas usage: 36789
  Block Number: 227
  Block Time: Tue May 15 2018 12:18:02 GMT-0400 (EDT)

eth_blockNumber
eth_accounts
eth_sendTransaction

  Transaction: 0x6ef29e50ec3e714a87219b31a35604ac39c9d35df0f6cd32920cc7984883788d
  Gas usage: 36789
  Block Number: 228
  Block Time: Tue May 15 2018 12:18:02 GMT-0400 (EDT)

eth_blockNumber
eth_accounts
eth_sendTransaction

  Transaction: 0x4e7736763bec0734be83a1360f9b87a74913ebca6253e04d80c2fe0e625c28d3
  Gas usage: 36789
  Block Number: 229
  Block Time: Tue May 15 2018 12:18:02 GMT-0400 (EDT)

eth_blockNumber
eth_accounts
eth_sendTransaction

  Transaction: 0x7312d3eb3a24d2b8fa7ec1847980c5d7eb3416f0b96ac39a2ccd8c7d215018f4
  Gas usage: 36789
  Block Number: 230
  Block Time: Tue May 15 2018 12:18:02 GMT-0400 (EDT)

eth_blockNumber
eth_accounts
eth_sendTransaction

  Transaction: 0x740438b41a0ba0c7f1f7e7fa7e5162f27985054ea7b3117f02aa3a518a0fb3b9
  Gas usage: 36789
  Block Number: 231
  Block Time: Tue May 15 2018 12:18:02 GMT-0400 (EDT)

eth_blockNumber
eth_blockNumber
eth_getBlockByNumber
eth_accounts
eth_call
eth_accounts
eth_call
```

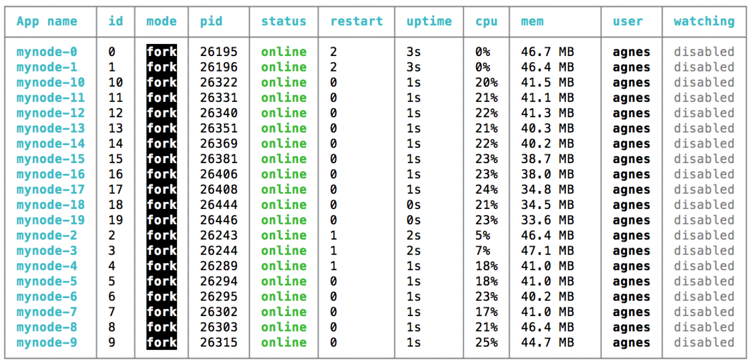
```
eth_blockNumber
eth_accounts
eth_sendTransaction

  Transaction: 0x0c2da3192465262a810c36aeae7173dde87d2d16d56896b98ce6d6b27a44c7c5
  Gas usage: 29695
  Block Number: 66
  Block Time: Tue May 15 2018 12:17:50 GMT-0400 (EDT)

eth_blockNumber
eth_accounts
eth_sendTransaction

  Transaction: 0xd20ade48c52d33c054852a91d6831ca4695132b0a0f206f25b4e0725c79ffbfa
  Gas usage: 29695
  Block Number: 67
  Block Time: Tue May 15 2018 12:17:50 GMT-0400 (EDT)

eth_blockNumber
eth_accounts
eth_sendTransaction

  Transaction: 0xe497eef6331bb3133bf5a3f320319f8bf340b4981a1bdce624b48abd5f57455a
  Gas usage: 29695
  Block Number: 68
  Block Time: Tue May 15 2018 12:17:50 GMT-0400 (EDT)

eth_blockNumber
eth_accounts
eth_sendTransaction

  Transaction: 0x06ccf69a2a5d7e1ed5c08cfc161f080836ae056026d774740cc282c59926eff6
  Gas usage: 29695
  Block Number: 69
  Block Time: Tue May 15 2018 12:17:50 GMT-0400 (EDT)

eth_blockNumber
eth_accounts
eth_sendTransaction

  Transaction: 0xb21dbf4c3549e8cd815327753075701eef591c741c162a5470c1270f3d9d3495
  Gas usage: 29695
  Block Number: 70
  Block Time: Tue May 15 2018 12:17:50 GMT-0400 (EDT)

eth_blockNumber
eth_accounts
eth_sendTransaction

  Transaction: 0x14d5041a5929ee48d21b5e81c168c8e4fa39218e5611af0f8dd44b1f0dc4c0b3
  Gas usage: 29695
  Block Number: 71
  Block Time: Tue May 15 2018 12:17:50 GMT-0400 (EDT)

eth_blockNumber
```

3.1 Limitations

This system was limited in that we were not able to tie each robot to a separate geth instance, due to the difficulty in configuring more than one geth node programatically (rather than through individual consoles). In the code used by [4], much of the extra complexity is associated with administering these nodes. Included in the repository is a forked and modified version of Strobel et. al's [4] C++ code to minimally launch and control multiple geth instances from the same console `server/geth_processes`. However, there was not time to bind the C++ to the Javascript backend, and as such this is left for potential future development.

Ultimately, the use-case for geth is to run nodes on the EVM, not to construct elaborate test-cases on a local machine. This limitation makes very experimental systems such as this very difficult to develop. In coming years, the development of more flexible and general multi-chain APIs will aid in developing these more specialised applications.

Figure 4: *Demonstration of running multiple geth processes on a single computer, using the eth-netstats command line visualiser*



```
App name    id   mode  pid    status  restart  uptime  cpu  mem      user   watching

mynode-0    0    fork  26195  online  2        3s      0%   46.7 MB  agnes  disabled
mynode-1    1    fork  26196  online  2        3s      0%   46.4 MB  agnes  disabled
mynode-10   10   fork  26322  online  0        1s      20%  41.5 MB  agnes  disabled
mynode-11   11   fork  26331  online  0        1s      21%  41.1 MB  agnes  disabled
mynode-12   12   fork  26340  online  0        1s      22%  41.3 MB  agnes  disabled
mynode-13   13   fork  26351  online  0        1s      21%  40.3 MB  agnes  disabled
mynode-14   14   fork  26369  online  0        1s      22%  40.2 MB  agnes  disabled
mynode-15   15   fork  26381  online  0        1s      23%  38.7 MB  agnes  disabled
mynode-16   16   fork  26406  online  0        1s      23%  38.0 MB  agnes  disabled
mynode-17   17   fork  26408  online  0        1s      24%  34.8 MB  agnes  disabled
mynode-18   18   fork  26444  online  0        0s      21%  34.5 MB  agnes  disabled
mynode-19   19   fork  26446  online  0        0s      23%  33.6 MB  agnes  disabled
mynode-2    2    fork  26243  online  1        2s      5%   46.4 MB  agnes  disabled
mynode-3    3    fork  26244  online  1        2s      7%   47.1 MB  agnes  disabled
mynode-4    4    fork  26289  online  1        1s      18%  41.0 MB  agnes  disabled
mynode-5    5    fork  26294  online  0        1s      18%  41.0 MB  agnes  disabled
mynode-6    6    fork  26295  online  0        1s      23%  40.2 MB  agnes  disabled
mynode-7    7    fork  26302  online  0        1s      17%  41.0 MB  agnes  disabled
mynode-8    8    fork  26303  online  0        1s      21%  46.4 MB  agnes  disabled
mynode-9    9    fork  26315  online  0        1s      25%  44.7 MB  agnes  disabled
```

3.2 Real World Considerations

A real world system would expect many issues not faced in our simulation. Many of these issues concern networking and sensing issues. For example, the robot nodes in a real life swarm may have much sparser neighborhoods of robots to communicate with. These networks would also be much more prone to failure and communication may take much longer in a real system, resulting in a potentially much slower convergence to consensus for a swarm.

In addition, we implement a Proof-of-Authority algorithm to submit blocks to the chain. Whilst PoA is fast, it relies on authentication to the system not being leaked, which immediately leaves it vulnerable to bad actors.

Overall the problems inherent in peer-to-peer robot systems are vital to several public and private entities, including shipping companies such as Amazon, and they mostly stem from the problems associated with most distributed systems.

3.3 Other Security Concerns

As mentioned previously, our scheme for identifying and blacklisting byzantine robots involves looking at the number of times a given robot has voted against the majority. With this scheme, it is easy to identify faulty robots that aren't voting properly as we expect them to do this consistently. In the event that a significant portion of the robots are instead adversarial though, it is possible that there exist strategies for subverting the security of the protocol.

For example, it is conceivable that a small group of adversarial robots could "gang-up" on non-Byzantine robots to influence them to repeatedly vote for the minority. The adversarial robots could then alter their votes as needed to remain undetected as Byzantine, while slowly reducing the numbers of non-Byzantine robots by making them appear Byzantine. A worst case scenario would be that these robots could weed out enough truthful robots to be able to mount a 51% attack.

While this idea is intriguing, this idea was not explored in the limited time of this study, but should be considered in future work that explores this technique.

3.4 Future Work

A main drawback of this implementation was that it was not possible in the time to create an individual geth node for each robot, thus limiting the peer-to-peer aspects of the system, and not allowing us to implement some of the Byzantine fault-tolerant logic discussed by [4].

In addition, we would like to extend this system to collect more extensive simulation data. A more flexible GUI that can allow people to setup arbitrary grids and swarms setups and perhaps allow them to modify a basic Solidity contract for each swarm would allow experiments to be set up and run considerably more efficiently. It would also be useful if we had time to integrate the database (which is already setup in our repo) into our experiment setup. This would allow us to easily store information about the entire state of the system, enabling more effective debugging and development.

We would also be interested in developing the following extensions:

1. Implementing heterogeneous consensus protocols among our swarms, for example implementing Federated Byzantine Agreement (as used by the Stellar protocol [17]), or a protocol that has more grounding in the physical system, such as Proof-of-Location [18]

2. Implementing consensus on a more complex set of data (something more complicated than a boolean or numerical value).

3. Using a different method of scaling for swarm robotics: for example, the Raiden network has a number of features that

4. Studying the behavior of adversarial actors instead of just faulty actors.

4 Conclusion

In this work, we discussed the application of multi-blockchain technology in Byzantine fault tolerant distributed consensus protocols for potential use in swarm robotics system, building off of the previous work of Strobel et al. While we were able to setup a Javascript web application for simulating a swarm-robotics based game successfully using two interacting blockchains, we were severely limited in testing our algorithm due to continually exceeding gas costs in the smart contracts used to implement the protocol. In many of the tests that we were able to run, the swarm robots were able to reach a consensus successfully, though occasionally at the cost of incorrectly identifying non-Byzantine robots as Byzantine.

As noted previously, this scheme is work towards a proof of concept and is not intended to be a fully implemented, fully vetted protocol. That being said, it is the authors' opinion that this system shows promise. For one, the Javascript web application is a useful frame work for future work on this problem. Furthermore, while we chose to use local development tools, this protocol could be adapted to run on a public Ethereum network. This opens up the potential for future work to overcome the gas limitations that hindered this study to choose reasonable parameters and protocol details such that the proposed protocol can work as intended using simple changes to the existing codebase.

References

[1] Saga, "SAGA – Swarm Robotics for Agricultural Applications," 2018.

[2] R. Bloss, "Advanced swarm robots addressing innovative tasks such as assembly, search, rescue, mapping, communication, aerial and other original applications," *Industrial Robot: An International Journal*, vol. 41, no. 5, pp. 408–412, 2014.

[3] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 3, pp. 382–401, 1982.

[4] V. Strobel, E. Castello Ferrer, and M. Dorigo, "Managing Byzantine Robots via Blockchain Technology in a Swarm Robotics Collective Decision Making Scenario," no. December 2017, p. 12, 2018.

[5] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," *www.Bitcoin.Org*, p. 9, 2008.

[6] G. Wood, "Ethereum: a secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, pp. 1–32, 2014.

[7] N. Szabo, "Smart Contracts," 1994.

[8] V. Buterin and V. Griffith, "Casper the Friendly Finality Gadget," pp. 1–10, 2017.

[9] V. Buterin, “A Proof of Stake Design Philosophy,” 2016.

[10] Statista, “Size of the Bitcoin blockchain from 2010 to 2017, by quarter (in megabytes),” 2017.

[11] Imbrex, “Sharding, Raiden, Plasma: The Scaling Solutions that Will Unchain Ethereum,” 2017.

[12] J. Stark, “Making Sense of Ethereum’s Layer 2 Scaling Solutions: State Channels, Plasma, and Truebit,” 2018.

[13] G. Wood, “Polkadot: Vision for a Heterogeneous Multi-Chain Framework,” pp. 1–21, 2017.

[14] J. Kwon and E. Buchman, “Cosmos: A Network of Distributed Ledgers,” 2018.

[15] G. Valentini, E. Ferrante, and M. Dorigo, “The Best-of-n Problem in Robot Swarms: Formalization, State of the Art, and Novel Perspectives,” *Frontiers in Robotics and AI*, vol. 4, no. March, 2017.

[16] A. Rosic, “What is Ethereum Gas: Step-by-Step Guide,” 2018.

[17] D. Mazières, “The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus,” pp. 1–45, 2015.

[18] F. Corp, “Whitepaper,” 2018.

Revised Results

Agnes Cameron, Bruno Prela, Mark Payne

1 Results

In general, we found that our model worked well for easier consensus problems (times where there was a large difference between the number of dirt and crop squares), but failed catastrophically when that difference was subtle. The time to consensus (measured as the time for the harvestBots to determine that the surveyBots had come to consensus), is shown in fig. 1. This largely decreases with the easiness of the consensus problem, as the robots are able to identify byzantine actors and reach global agreement faster. The results of this figure exclude simulations that failed to reach a consensus, however: the chance of these events occurring decreased as the problem got easier. For all these simulations, 1/3 of the actors was Byzantine, as in the original Byzantine Generals Problem.

Figure 2 shows the success in identifying Byzantine robots as a function of the easiness. In easy problems, where most of the robots agree immediately, the discovery of Byzantine actors is successful. However, for harder problems, the uncertainty in the swarm as a whole makes it much harder to identify Byzantine actors based purely on the votes that they gave. Figure 3 shows the number of false accusations as a function of easiness. In the harder problems, robots are more likely to vote for the 'wrong' option, making them vulnerable, based on their vote pattern, to being identified as Byzantine.

These harder problems sometimes resulted in catastrophic failure, where the false accusation of an honest robot makes the number of honest votes decrease, making it easier for Byzantine actors to succeed.

2 Brief Discussion

Our system was able to reach consensus between two blockchains in the face of Byzantine actors, fulfilling a key goal of the project. However, our Byzantine detection algorithm is sorely in need of improvement, and the system cannot be described as 'Byzantine Fault Tolerant', as in Lamport's original problem, if there is any majority at all between the honest actors, the correct vote should be made. In addition, even in 'successful' runs, the occasional false accusation of an honest robot could be costly to a system, and unacceptable in an

Figure 1: Average time to consensus as a function of easiness

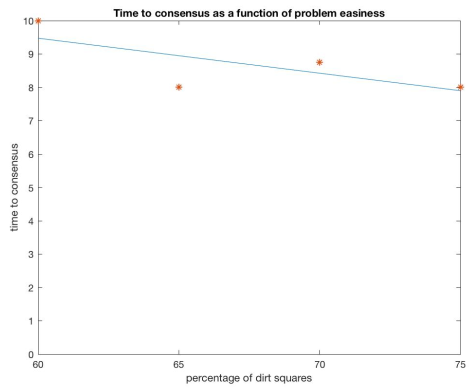
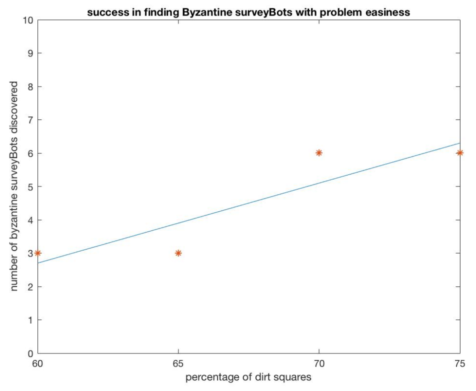
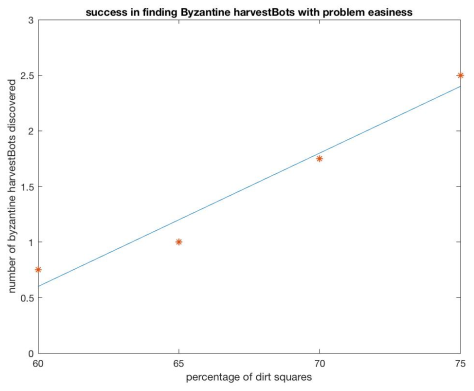


Figure 2: Average number of Byzantine robots successfully found, as a function of easiness

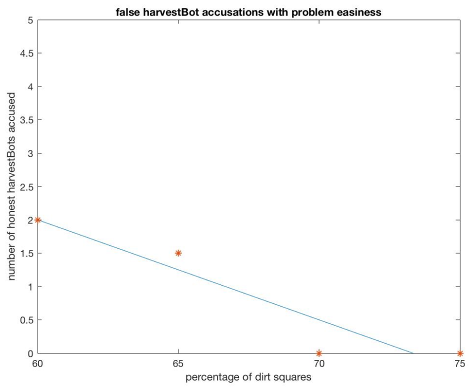


(a) *Average byzantine surveyBots found (out of 6 possible)*

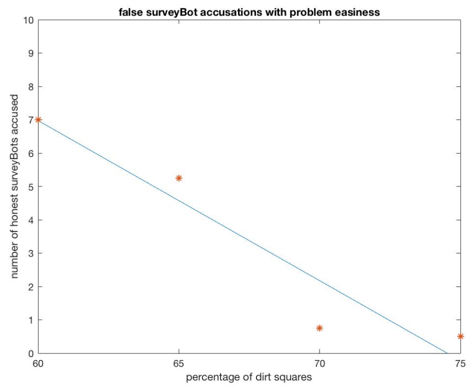


(b) *Average byzantine harvestBots found (out of 3 possible)*

Figure 3: Average number of honest robots falsely accused, as a function of easiness

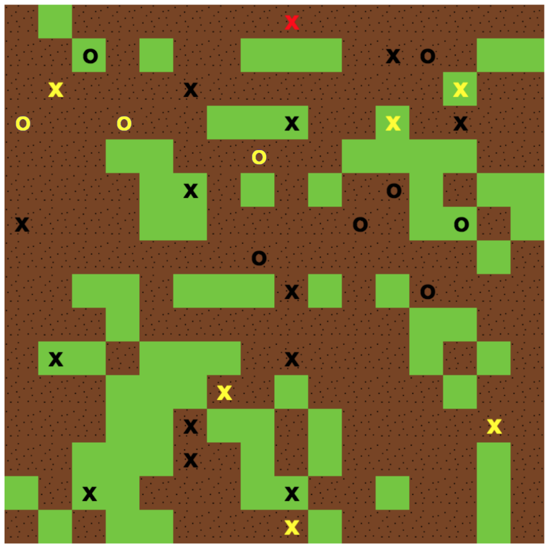


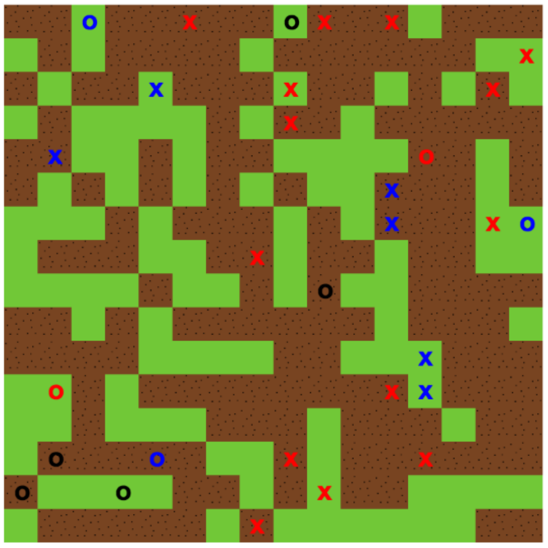
(a) *Average honest surveyBots falsely accused (out of 14 possible)*



(b) *Average honest harvestBots falsely accused (out of 14 possible)*

Figure 4: Comparing different outcomes for the swarm


(a) *Typical successful round, 70% dirt. Successfully-identified Byzantine robots are in yellow, undiscovered Byzantine in blue, and falsely accused in red*


(b) *Catastrophic failure , 60% dirt. Successfully-identified Byzantine robots are in yellow, undiscovered Byzantine in blue, and falsely accused in red*

industrial context. By improving the Byzantine algorithm, we would hope that this can become a workable system.

There was also an additional but subtler issue associated with the spatial distribution of the harvestBots. Sometimes, identification of the byzantine robots failed as the harvestBots were too sparsely distributed. Without the opportunity to add robots to a peer network, it was harder to identify a global consensus. Were we to run these experiments another time, we would make the range of the harvestBots larger to compensate for this sparse distribution. However, this does confirm the importance of the peer voting mechanism used.