The Blockchain Anomaly

Christopher Natoli
NICTA/Data61-CSIRO
University of Sydney
cnat5672@uni.sydney.edu.au

Vincent Gramoli
NICTA/Data61-CSIRO
University of Sydney
vincent.gramoli@sydney.edu.au

Abstract

Most popular blockchain solutions, like Bitcoin, rely on proof-of-work, guaranteeing that the output of the consensus is agreed upon with high probability. However, this probability depends on the delivery of messages and that the computational power of the system is sufficiently scattered among pools of nodes in the network so that no pool can mine more blocks faster than the crowd. New approaches, like Ethereum, generalise the proof-of-work approach by letting individuals deploy their own private blockchain with high transaction throughput. As companies are starting to deploy private chains, it has become crucial to better understand the guarantees blockchains offer in such a small and controlled environment.

In this paper, we present the *Blockchain Anomaly*, an execution that we experienced when building our private chain at NICTA/Data61. Even though this anomaly has never been acknowledged before, it may translate into dramatic consequences for the user of blockchains. Named after the infamous Paxos anomaly, this anomaly makes dependent transactions, like "Bob sends money to Carole after he received money from Alice" impossible. This anomaly relies on the fact that existing blockchains do not ensure consensus safety deterministically: there is no way for Bob to make sure that Alice actually sent him coins without Bob using an external mechanism, like converting these coins into a fiat currency that allows him to withdraw. We also explore smart contracts as a potential alternative to transactions in order to freeze coins, and show implementations of smart contract that can suffer from the Blockchain anomaly and others that may cope with it.

Keywords: Ethereum; Paxos anomaly; private chain; smart contract

1 Introduction

Mainstream public blockchain systems, like Bitcoin [16] and Ethereum [20], require to reach consensus on Internet despite the presence of malicious participants. Yet, it is impossible for a distributed system including a faulty participant to reach consensus if messages may not be delivered within a bounded time [11]. This contradiction raises interesting research questions regarding the formal properties that are sacrificed in these blockchain systems. Foundational consensus algorithms [7] were proposed to never reach a decision in case of arbitrary message delays, but to respond only correctly if ever. Surprisingly, these blockchain systems adopt a different approach, sometimes responding incorrectly. These few last years, the concept of *private chain* gained traction for its ability to offer blockchain among multiple companies in a private, controlled environment. Three months ago, eleven banks collaborated successfully in deploying an Ethereum private chain to perform transactions across North America, Europe and Asia.[1] To understand the limitations of consensus and its potential consequences in the context of private chains, we deployed our own private chain and stress-tested the systems in corner-case situations.

In this paper, we present the *Blockchain anomaly*, a new problem named after the Paxos anomaly [14, 1, 2, 13], that prevents Bob from executing a transaction based on the current state of the blockchain. In particular, we identified a complex scenario where the agreement on the state of the blockchain is not sufficient to guarantee immutability of the chain. This anomaly can lead to dramatic consequences, like the loss of virtual assets or a double-spending attack. We also show that some *smart contracts*, expressive code snippets that help defining how virtual assets can be owned and exchanged in the system, may suffer from the Blockchain anomaly. Our results outline the risk of using a blockchain in a private context without understanding its complex design features. We terminate our experience report by providing the source code of a more complex smart contract that can circumvent a particular example of the Blockchain anomaly.

Most blockchain systems track a transaction by including it in a block that gets mined before being appended to the chain of existing blocks, hence called *blockchain*. The consensus algorithm guarantees a total order on these blocks, so that the chain does not end up being a tree. This process is actually executed speculatively in that multiple new blocks can be appended transiently to the last block of the chain—a transient branching process known as a *fork*. Once the fork is discovered, meaning that the participants learn about the two branches, the longest branch is adopted as the valid one. Blockchain systems usually assume that forks can grow up to some limited depth, as extending a branch requires to solve a complex challenge that boils down to spending a long time during which one gets likely notified of the longest chain. Bitcoin recommends six blocks to be mined after a transaction is issued to consider the transaction accepted by the system. Similarly, Ethereum states that five to eleven more blocks should be appended after a block for it to be accepted [20].

However, consensus cannot be solved in the general case. In particular, foundational results of distributed computing indicate that consensus cannot be reached if there is no upper-bound on the time for a message to be delivered and if some participant may fail [11]. Consensus is usually expressed with three properties: *agreement* indicating that if two non-faulty participants decide they decide on the same block, *validity* indicating that the decided block should be one of the blocks that were proposed and *termination* indicating that eventually a correct participant decides. The common decision that is taken by famous consensus protocols, like Paxos [15] and Raft [17], is to make sure that if the messages get delayed, at least validity and agreement remain ensured. This is achieved by having the algorithm doing nothing in the worst case, hence sacrificing termination to ensure that only correct responses—satisfying both validity and agreement—can be returned. These consensus algorithms are appealing, because if after some time the network stabilises and messages get delivered in a bounded time, then consensus is reached [7].

We illustrate the Blockchain anomaly and describe a distributed execution where even committed transactions of a private chain get reordered so that the latest transaction ends up being committed first. We chose Ethereum for our experiments as it is a mainstream blockchain system that allows the deployment of private chains. We show how to reproduce the Blockchain anomaly by following the same execution, where messages get delayed between machines while some miner mines new blocks. Despite transactions being already committed the eventual delivery of messages produces a reorganisation reordering some of the committed transactions. In our execution, miners are setup to dedicate different number of cores to the mining process, hence mining at different speeds. We argue that the misconfiguration of a machine and the heterogeneous mining capabilities of machines belonging to different companies are sufficiently realistic to allow an attacker to execute a double-spending attack. Finally, we discuss the relations of the Blockchain anomaly to other problems and observe that it is not confined to the Ethereum blockchain but could potentially apply to proof-of-stake private blockchains as well, requiring further investigations.

Section 2 overviews the blockchain technology, the Paxos anomaly and defines the important terms of the paper. In Section 3, we present the blockchain anomaly. In Section 4, we present our experiments based on an Ethereum private chain. In Section 5, we explain how replacing transactions by smart contracts could help bypassing the anomaly. Section 6 discusses the Blockchain anomaly in other settings. Section 7 presents the related work. And Section 8 concludes.

[1] BMO Financial Group, Credit Suisse, CBA, HSBC, Natixis, Royal Bank of Scotland, TD Bank, UBS, UniCredit and Wells Fargo as explained at http://www.ibtimes.co.uk/r3-connects-11-banks-distributed-ledger-using-ethereum-microsoft-azure-1539044.

2 Preliminaries

In this section, we present the key concepts of Bitcoin and Ethereum consensus protocols, the condition of their termination and the Paxos anomaly before presenting the general model.

2.1 Blockchain Systems

A blockchain can be considered as a replicated state machine [22] where a reversed link between blocks is a pointer from a state to its preceding state as depicted in Figure 1. Consensus is necessary to totally order the blocks, hence maintaining the chain structure. To reach consensus despite arbitrary failures, including malicious behaviors, traditional blockchain systems adopted a technique based on proof-of-work, requiring a proof of computation [8]. Specialised peers, called *miners*, provably solve a hashcash crypto puzzle [3] before a new block can be appended to the blockchain. Given a block and a threshold, a miner repeatedly selects a nonce and applies a pseudo-random function to this block and the selected nonce until it obtains a result lower than the threshold. The difficulty of this work limits the rate at which new blocks can be generated by the network.

2.2 From Nakamoto's Consensus to Smart Contracts

Nakamoto's consensus [16] is at the core of Bitcoin, the mainstream decentralised digital currency. Interestingly, Nakamoto's consensus does not guarantee agreement deterministically. Instead it guarantees that agreement is met with some probability close to 1. The difficulty of the crypto puzzles used in Bitcoin leads to mining a block every 10 minutes. The advantage of this long period, is that it is relatively rare for the blockchain to *fork* due to blocks being simultaneously mined and Bitcoin resolves these forks by choosing the longest branch and discarding the other(s).

Ethereum [21] is a recent open source cryptocurrency platform that also builds upon proof-of-work. As opposed to Bitcoin's consensus protocol, Ethereum generates one block every 12–15 seconds. While it improves the throughput (transactions per second) it also favours transient forks as miners are more likely to propose new blocks simultaneously. To avoid frequently wasting mining efforts to resolve forks, Ethereum uses the GHOST (Greedy Heaviest Observed Subtree) protocol that does not necessarily discards all the, so called uncle, blocks of non selected branches. Ethereum offers a Turing-complete programming language that can be used to write *smart contracts* [18] that define new ownership rules.

2.3 Termination of Consensus

By relaxing the agreement property of consensus, blockchain systems can guarantee termination deterministically. In the context of blockchain, termination of consensus indicates

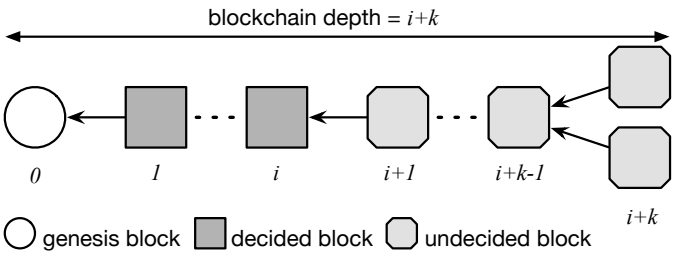

Figure 1: The blockchain structure starts with a genesis block at index 0 and links successive blocks in reverse order of their index; a new block is decided at index $i > 0$ when the blockchain depth reaches $i + k$ (note that a blockchain of depth 0 is a genesis block)

that a block has been *decided* for the next available block index. We say that all the transactions of a decided block are *committed*.[2] This decision upon a block inclusion in the chain is necessary for cryptocurrency exchange platforms, for example, to determine that coins of a particular type that are newly minted[3] within this block can be converted into *altcoins* (coins of a different type) or fiat currencies (e.g., EUR, USD). In particular, observing that a block was mined and appended to the chain is not sufficient to guarantee that it is decided: this block could be part of one branch of a transient fork without consensus being reached yet on any of these branches.

Figure 1 depicts the termination of consensus on the index i of a blockchain. An arrow pointing from left to right indicates that a block contains a hash of its predecessor block, the one located immediately on its left. Newly mined blocks are added to the right end of the blockchain that may fork transiently if multiple blocks referring to the same predecessor get mined concurrently. Forks are only transient and their resolution depends on the blockchain system in use. The consensus for an index i terminates when participants decide on the new block to be assigned at index i. The decision upon the block at index i occurs for all $i > 0$ when the blockchain depth reaches $i + k$, where $k \geq 0$ is a constant dependent on the Blockchain system.

Different blockchain systems adopt different values of k to define termination. In Bitcoin (btc), $k_{btc} = 5$, meaning that the block at index i is decided—consensus for index i terminates—when the $k_{btc} + 1 = 6$ blocks at indices $i, ..., i + 5$ have been successfully mined. As we previously mentioned, a new block is decided every 10 minutes in Bitcoin, hence it takes $(k_{btc} + 1) * 10\,min = 1\,hour$ for a transaction to be committed in Bitcoin. In Ethereum (eth) since verion 1.3.5 Homestead, $k_{eth} = 11$, meaning that the block at index i is decided—consensus for index i terminates—when the blockchain depth reaches $i + 11$. Hence it takes

[2] Here, we use the term "committed" rather than "confirmed" as, in the blockchain terminology, a transaction is meant to be "confirmed" sometimes when only its block is mined, and sometimes when $k + 1$ blocks get mined (its own block and the k successor blocks).

[3] As opposed to *mining* that includes the computation of the miners, *minting* consists simply of the creation of coins.

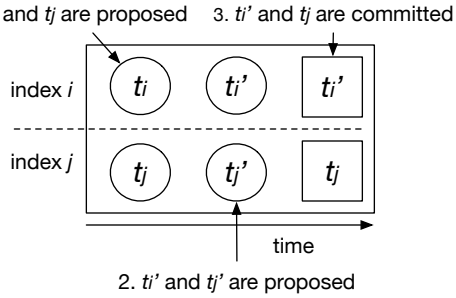

Figure 2: The Paxos anomaly: a first leader proposes t_i and t_j for slots i and $j > i$ (with t_j being implicitly conditional to the commit of t_i), a second leader proposes t'_i and t'_j while a third leader commits t'_i and t_j for slots i and j, respectively, hence violating the dependency between t_i and t_j

$(k_{eth} + 1) * 15\,sec = 3\,min$ for transactions to be committed in Ethereum. Note that some cryptocurrency exchange platforms adopt different values of k to adjust the probability of agreement, hence QuadrigaCX Ether Trading waits for $k'_{btc} + 1 = 4$ blocks to be mined in the Bitcoin blockchain while it waits for $k_{eth} + 1 = 12$ blocks to be mined in the Ethereum blockchain.[4]

2.4 The Paxos Anomaly

Paxos is a famous consensus protocol originally guaranteeing agreement and validity despite crash failures [15]. The *Paxos anomaly* [1, 14] stems from the difficulty of implementing conditional requests (or transactions) in Paxos: Paxos decides on individual proposed transactions, potentially violating dependencies between transactions even when proposed by the same requester as depicted in Figure 2 where a slot can be viewed as the index of the decision. These dependencies can be useful to make the execution of a transaction t_j dependent on the successful execution of a previous transaction t_i: for example if Bob wants to transfer an amount of money to Carole (t_j) only if he successfully received some money from Alice (t_i). In centralised systems, this anomaly can be easily avoided by enforcing an ordering on these transactions by simply forwarding all requests to a primary node or coordinator [14]. In Paxos as in fully decentralised systems, however, the first transaction may not be decided in favour of another proposed transaction in a first consensus instance while in a subsequent consensus instance the second transaction may be successfully decided. This results in a violation of the condition that the second transaction should be decided only if the first transaction was decided.

Below we present the Blockchain anomaly due to the decentralised aspects of blockchain systems, like Bitcoin and Ethereum. The Blockchain anomaly shares similarities with the Paxos anomaly, except that it can occur when transactions, issued by different nodes of the system, are not even concurrent.

[4] https://www.quadrigacx.com/faq.

2.5 General Model

We consider a distributed blockchain system of n peers where peers can exchange coins from one to another through *transactions*. The goal is for the system to implement a *ledger* abstraction as a public permanent and auditable records of all transactions. The ledger is implemented with a *blockchain*, a series of transaction blocks, starting with a special block called the *genesis block*. Blocks are singly linked one after another up to the genesis block—each non-genesis block containing a hash of the previously accepted block—and define the current state of the ledger as the set of transactions that ever occurred. The block index or *slot* increases monotonically from the index 0 of the genesis block.

Peers can fail arbitrarily, they can stop working and can be malicious. Any peer can issue transactions that get recorded into the *transaction pool*. Only miners can bundle a subset of the pool of transactions into a block after ensuring that there are sufficient funds available on the accounts of the ledger and that these transactions do not conflict. The system uses consensus to guarantee that no malicious peers are trying to double-spend some coins by issuing two conflicting transactions concurrently to different miners. To this end, we consider a consensus protocol based on proof-of-work so that miners bundle transactions from the transaction pool into a block by solving a crypto puzzle in exchange of coins for the system to decide on the new block. The *difficulty* of the crypto puzzle determines the rate at which new blocks can be mined: the higher the difficulty, the slower the rate.

At times though the blockchain may fork transiently[5], indicating that multiple blocks were appended to a unique block, in which case conflicting transactions could potentially be part of blocks on different branches of the chain. If this fork happens, then a *reorganisation* process eventually occurs to resolve the fork by uniquely identifying one of the two branches as the right one.

3 The Blockchain Anomaly

We present the Blockchain anomaly, an anomaly that affects mainstream blockchain systems whose consensus protocol does not ensures agreement deterministically.

3.1 Causes of the Blockchain Anomaly

The problem stems from the asynchrony of the network, in which message delays cannot be bounded, and the termination of consensus. Although two miners mine on the same chain starting from the same genesis block, a long enough delay in messages between them could lead to having the miners seemingly agree separately on different branches containing more than k blocks each, for any k. This anomaly is dramatic as it can lead to simple attacks within any private network where users have an incentive to maximise their profits—in terms of coins, stock options or arbitrary ownership. Moreover, this scenario is realistic in the context of

[5] This transient forks are sometimes referred to as *ephemeral forks* as they are expected in the normal course of the execution.

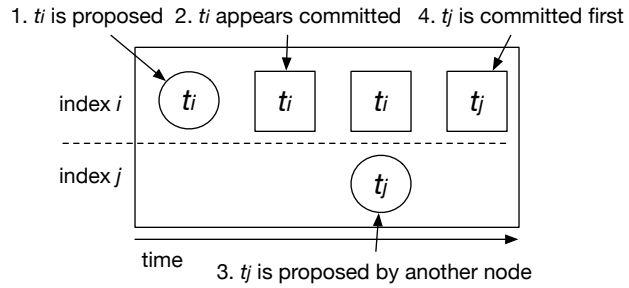

Figure 3: The Blockchain anomaly: a first client issues t_i that gets successfully mined and committed then a second client issues t_j, with t_j being conditional to the commit of t_i (note that $j \geq i + k$ for t_i to be committed before t_j gets issued), but the transaction t_j gets finally reorganised and successfully committed before t_i, hence violating the dependency between t_i and t_j

private chain where the employees of a company, like NICTA/Data61, have direct access to some of the network resources. When messages get finally delivered, the results of the disagreement creates inconsistencies.

3.2 Uncommitting Transactions

Figure 3 depicts the Blockchain anomaly, where a transaction t_i gets committed as part of slot i from the standpoint of some nodes. Based on this observation, one proposes a new transaction t_j knowing that t_i was successfully committed. Again, one can imagine a simple scenario where "Bob transfers an amount of money to Carole" (t_j) only if "Bob had successfully received some money from Alice" (t_i) before. However, once these nodes get notified of another branch of committed transactions, they decide to reorganise the branch to resolve the fork. The reorganisation removes the committed transaction t_i from slot i. Later, the transaction t_j is successfully committed in slot i.

The anomaly stems from the violation of the dependency between t_j and t_i: t_j occurred meaning that Bob has transferred an amount of money to Carole, however, t_i did not occur meaning that Bob did not receive money from Alice. Note that in Bitcoin, transaction t_i gets discarded whereas in Ethereum transaction t_i may in some cases be committed in slot j.

3.3 Facilitating a Double-Spending Attack

One dramatic consequence of the Blockchain anomaly is the possibility for an attacker to execute a *double-spending* attack: converting, for example, all his coins into goods twice. The scenario consists of the attacker issuing a first transaction t_1 that converts all its coins into goods in block i and starting mining blocks after block $i - 1$ in isolation of the network. As part of this mining, the attacker mines another transaction t_2 that also converts all its coins into goods. The attacker then waits for the blockchain depth to reach $i + k$ after what it can collect its goods as a result of transaction t_1, then it publicises its longer chain without t_1 so that the chain gets adopted by the rest of network. t_2 gets committed in block j and after the chain depth reaches $j + k$, the peer can collect its goods for the second time. Note that even if one tries to recommit t_1 later, the transaction will be invalidated because the balance is insufficient, however, the double-spending already occurred.

3.4 Tracking Blockchain Anomalies

Another dramatic aspect of the Blockchain anomaly is that it goes undetected. More specifically, the Blockchain anomaly relies on a wrongly committed state of the blockchain. Once the wrongly committed state gets uncommitted, there is no way to a posteriori observe this problematic state and to notice that a blockchain anomaly occurred. Although it is possible to observe that a peer mined several blocks in a row, there is no way to track down the beneficiaries of the Blockchain anomaly. This dangerously incentivises participants of the private chain to leverage the Blockchain anomaly to attack the chain.

4 Experimental Evaluation

In this section, we describe a distributed execution involving a private chain that results in the Blockchain anomaly.

4.1 Experimental Setup

We deployed a private blockchain system in our local area network using geth version 1.4.0, which is a Go implementation of the command line interface for running an Ethereum node. We setup three machines connected through a 1 Gbps network, two consisting of miners, p_1 and p_3, generating blocks and one consisting of a peer p_2 simply submitting transactions. Peers p_1 and p_2 consist of 2 machines with 4 $\times$ AMD Opteron 6378 16-core CPU running at 2.40 GHz with 512 GB DDR3 RAM, each. Peer p_3 consists of a machine with 2 $\times$ 6-core Intel Xeon E5-260 running at 2.1 GHz with 32 GB DDR3 RAM.

We artificially created a network delay by transiently annihilating connection points between machines. Note that such artificial delays could be reproduced by simply unplugging an ethernet cable connecting a computer to the company network and does not require an employee to access physically a switch room.

Also, we made sure p_3 would mine faster than p_1, by mining with the 24 hardware threads of p_3 and a single hardware thread of p_1. The same speed difference could be obtained between a loaded server and a server that does run any other service besides mining. Note that hardware characteristics may also help one machine mine faster than the rest of a private chain network. For example, a machine equipped with an AMD Radeon R9 290X would mine faster in Ethereum than a pool of 25 machines, each of them mining with an Intel Core i7.

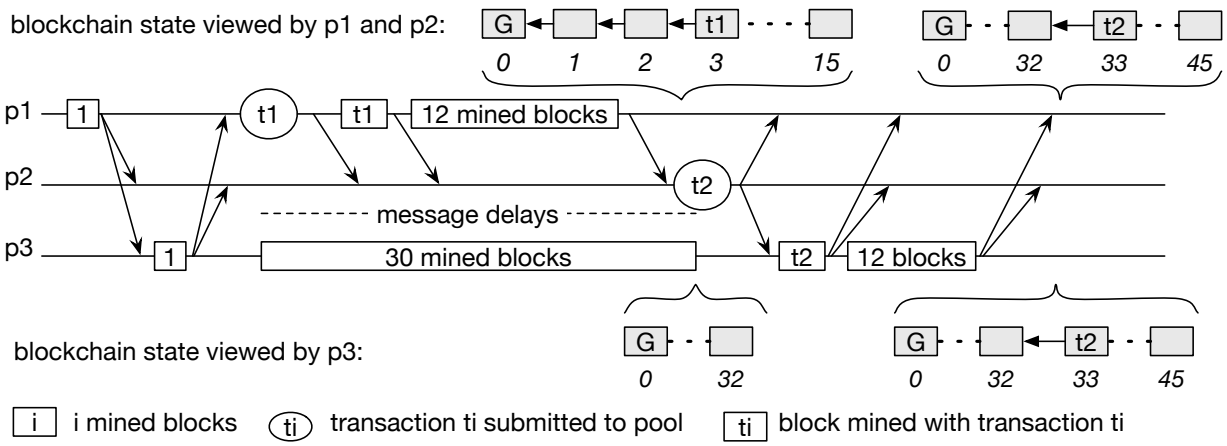

Figure 4: Execution scenario leading to the Blockchain anomaly: p_3 mines a longer chain than p_1 without including t_1 and without disseminating new blocks until it forces a reorganisation that imposes t_2 to be committed while t_1 appears finally uncommitted

4.2 Distributed Execution

The default case of the anomaly occurs with a conditional transaction. A peer in the system has a condition that it will only send money if some other peers transferred him some coins successfully. As mentioned previously, for the transaction to be committed, there must be at least k blocks mined after the block containing the transaction. In our experiment, the client only sends coins once the peer owns a verified amount of coins. The peer performs a transaction t_2 only if it was shown by the system that the previous transaction t_1 had been committed and the money was successfully transferred to its wallet.

Figure 4 depicts the distributed execution leading to the Blockchain anomaly where p_1, p_2 and p_3 exchange information about the blockchain whose genesis block is denoted 'G'.

1. Peer p_1 mines a first block after the genesis block and informs p_2 and p_3 to update their view of the blockchain state.
2. Peer p_3 mines a second block and informs p_1 and p_2 of this new block.
3. A network delay is introduced between peers p_1 and p_2 on the one hand, and peer p_3 on the other hand.
4. Peer p_1 submits transaction t_1 and informs p_2 but fails to inform p_3 due to the network delay. In the meantime, peer p_3 starts mining a long series of 30 blocks.
5. Peer p_1 mines a block that includes transaction t_1 and mines 12 subsequent blocks; p_1 then informs p_2 but not p_3 due to the network delay.
6. Peer p_2 receives the notification from p_1 that t_1 is committed because its block and k subsequent blocks are mined; then p_2 decides to submit transaction t_2 that should only execute after t_1.
7. The network becomes responsive and p_3 who receives the information that t_2 is submitted, mined t_2 in a block along with 12 subsequent blocks.
8. Once peers p_1 and p_2 receive from p_3 the longest chain of 45 blocks, they adopt this chain, discarding or postponing the blocks that were at indices 2 to 15, including the transaction t_1, of their chain.
9. All peers agree on the final chain of 45 blocks in which t_2 is committed and where t_1 is finally not committed before t_2.

This execution results in a violation of the conditional property of transaction t_2 stating that t_2 should only execute if t_1 executed first. This violation occurred because transaction t_1 had been included in one chain, decided and agreed by two of the participants, it was then changed after the message of the third participant was finally delivered to the rest of the network.

4.3 Automating the Reproduction of the Anomaly

To illustrate the anomaly, we wrote a script that automated the execution depicted in Figure 4. Figure 5 represents the execution of a script that execute 8 iterations of the Blockchain anomaly over a period of 50 minutes. Again the

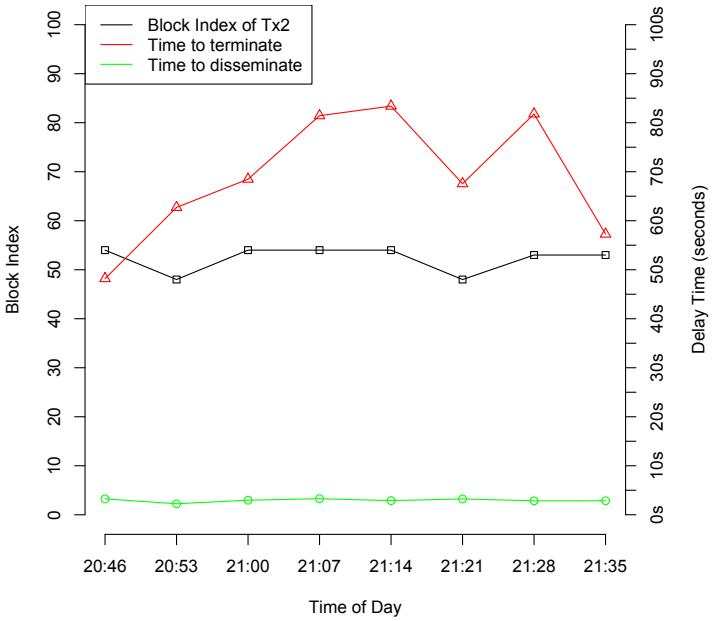

Figure 5: Automated executions of the Blockchain anomalies over a period of 50 *min*, the execution is non-determinstic due to the randomness of the mining process and the network delay between peers

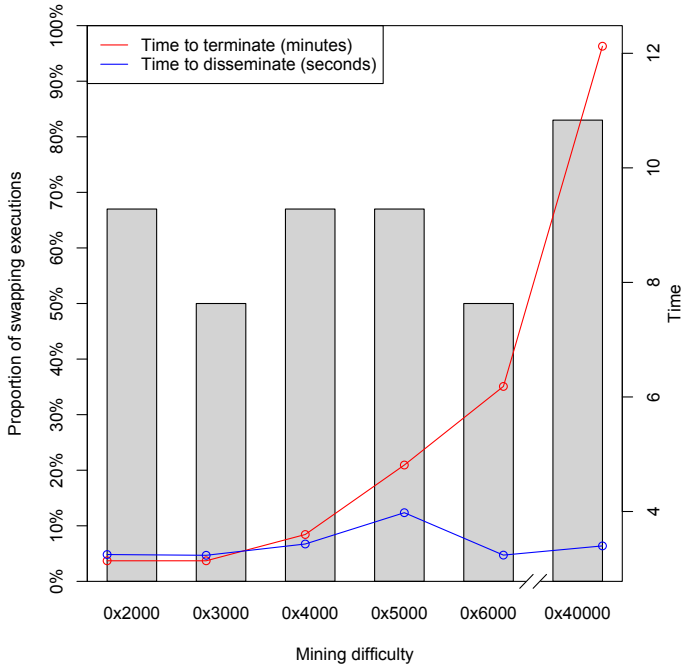

Figure 6: The proportion of transaction swaps observed does not depend on the difficulty, as opposed to the consensus termination that increases with the difficulty

goal is to wait until t_1 gets committed before issuing t_2 that ends up being committed while t_1 does not appear to be. Note that this is similar to Figure 3 except that t_2 is not necessarily included at the index t_1 occupied initially. In particular, the block in which t_2 gets included varies from one iteration to another due to the non-determinism of the execution as indicated by the curve with square points. This non-determinism is explained by the randomness of the mining process and the latency of the network that also impacts the time it takes for the consensus to terminate (curve with triangle points) in each iteration of the experiment. Note that we use $k = 11$ in this experiment, making sure that 12 blocks were successfully mined, as recommended since the release of Ethereum 1.3.5 Homestead, for the consensus to terminate.

As expected, in each of these eight cases we observed the Blockchain anomaly: even though t_2 was issued after t_1 was successfully observed as committed, if the messages get successfully delivered, then the reorganisation results in t_2 being committed while t_1 is not. Finally, we can observe that the time to disseminate a committed transaction to all the peers of the network is much shorter than the termination delay. This is due to the time needed to mine a block, which is significantly larger than the latency of our network.

4.4 Swap Frequency with Different Mining Difficulties

In the previous experiment, we used the default Ethereum difficulty (0x400) and automated the execution with a precise script. To better understand the cause of the anomaly we tried reproducing the anomaly by hand (without the script) with larger difficulties.

Figure 6 depicts the average number of blockchain anomalies leading to a swap (where both t_2 and t_1 are eventually committed in reverse order) occurring in our private chain for 6 different mining difficulties. Each bar results from the average number of anomalies observed during 6 manual runs of the scenario depicted in Figure 4. More precisely, the figure reports the *swap* scenarios, where the first transaction t_1 gets successfully committed before t_2 gets issued, and eventually transactions t_1 and t_2 appear committed in reverse order.

We ran this particular experiment with $k = 10$ for the termination of consensus, meaning that t_1 was mined in block at index i and it was committed once the chain depth reached $i + 10$ blocks. (We presented the anomaly in the case where $k = 11$ in Section 4.3.) At first, we thought that the occurrence of this Blockchain anomaly was dependent on the difficulty of mining a block: the faster a block could be mined, the more likely the anomaly would occur. To validate this, we varied the mining difficulties from 0x2000 to 0x40000 and measure the frequency of the Blockchain anomaly over 6 executions for each difficulty value. We observe that there was no significant correlation between the difficulty and the occurrence of the anomaly and that in average we could observe the swap 6 times out of 10.

We also measured the time it would take for consensus to terminate in these scenarios (upper curve) and observed, as expected, that the termination time was proportional to the difficulty. This is explained by the fact that the difficulty impacts the time needed to mine a block, which in turn, impacts the time it takes to mine $k + 1$ blocks for termination.

```
1  contract conditionalPayment {
2
3    uint32 paid; // to keep track of the amount paid by Alice when deciding on Bob's transfer
4    mapping (address => uint256) public balances; // map addresses to their respective balance
5    address A = 0x57ec7927841e2d25aad5f335e3b701369b177392; // the address of Alice's account
6    address B = 0x5ae58375c89896b09045de349289af9034902905; // the address of Bob's account
7
8    modifier onlyFrom(address _address) { // enables execution of functions depending on invoker
9      if (msg.sender != _address) throw;
10     _
11   }
12
13   function sendTo(address B, uint32 _amount) onlyFrom(A) { // Alice sends money to Bob
14     if (balances[A] >= _amount) { // checking the sufficiency of funds available
15       balances[A] -= _amount;
16       balances[B] += _amount;
17       paid = _amount; // sorting the amount paid
18     }
19   }
20
21   function sendIfReceived(address C, uint32 _amount) onlyFrom(B) { // Bob sends money to Carole
22     if (paid > _amount) { // only if the previous payment was sufficient
23       balances[B] -= _amount;
24       balances[C] += _amount;
25     } else {
26       throw; // cancel contract execution
27     }
28   }
29 }
```

Figure 7: A smart contract written in the Solidity programming language to replace transactions prone to the blockchain anomaly: the `sendIfReceived` function checks that the transfer from A to B occurred before executing the transfer from B to C

In addition we report the time it would take for a transaction in a mined block to be disseminated to all the peers of the network (bottom curve) and observed that it was not related to the difficulty. Finally, we observed that having a network delay greater than the time to mine was foundational to the observation of the anomaly.

5 Smart Contracts

Smart contracts are a foundational aspect of the *Ethereum* system, as they are distributed code execution based on conditional aspects. The contracts can be programmed to allow for certain conditions to be met in order for the code to be executed. What we found was that the anomaly prevention depended entirely on the programming of the smart contract. This means that if a smart contract was coded so that it did not properly check the condition that the first transaction had occurred, it would execute as normal, acting like a normal transaction and suffering from the anomaly.

5.1 On-Chain vs. Off-Chain Computation

In Figure 7, we illustrate the writing of a smart contract in the Solidity programming language with which we could not observe the anomaly. The key point is that the `sendIfReceived` function groups two steps: the check that the amount has been paid at Line 22 and the payment that results from this successful check at Lines 23 and 24. Because these two steps are executed on-chain, we know that one has to be necessarily true for the second to occur.

However, if the two steps were parts of two separate functions of the contract, one checking that the amount had been paid and another that would do the payment and be invoked upon the returned value of the former then the anomaly could arise. For example, consider Figure 8 where one function, `checkPayment`, checks that the payment from Alice proceeded correctly (Lines 3–7) and the other function, `sendIfReceived`, is modified to execute the payment unconditionally (Lines 9–13). Even if Bob invokes `checkPayment` and observes that it returns successfully before invoking `sendIfReceived` the anomaly may arise. The reason is that the check is made off-chain and nothing guarantees that the payment from Alice was not reorganised while Bob was checking the result off-line.

To conclude, it looks like the former contract in Figure 7 has higher chances of not suffering from the Blockchain anomaly than the smart contract of Figure 8 as it executes the check and the conditional transfer on-chain, however, this does not guarantee that the smart contract of Figure 7 is immune to the blockchain anomaly. Further investigation is needed to prove it formally. In addition and just like transactions, smart contracts must be included in a block that gets mined and appended to the blockchain. Its inclusion into the blockchain even with k subsequent blocks may suffer from a reordering as well, and lead to other kind of anomalies.

```
contract problematicConditionalPayment {
  ...
  function checkPayment(address B, uint32 _amount) onlyFrom(B) constant returns (bool result) {
    if (paid > _amount) { // check that Alice paid
      return true;
    } else throw;
  }

  function sendIfReceived(address C, uint32 _amount) onlyFrom(B) { // Bob sends money to Carole
    balances[B] -= _amount;
    balances[C] += _amount;
  }
}
```

Figure 8: Executing the transfer to Carole in a separate function may suffer from the Blockchain anomaly

5.2 Multisignatures and the Case of Bitcoin

Even without the Turing-complete scripting language, there may be ways in Bitcoin to bypass the Blockchain anomaly. The idea is to change a conditional transaction into a joint payment that includes both the conditional transaction and the action enabling its condition. The idea of including the transaction and the action is similar to the idea of grouping in the same contract function `SendIfReceived` of Figure 7, the check and the transfer that we described before.

The joint payment will represent the payment of Carole by both Alice and Bob. The payment will thus take two inputs, owned by different people, and give one output. Because the coins of these two inputs come from different addresses, the joint payment needs two different signatures. The joint payment can be achieved with a `multisig` transaction in Bitcoin so that the `multisig` transaction requires either two signatures from Alice, Bob and an arbitrator, Donald, in order to execute. If both Alice and Bob sign the transaction, then it executes and Carole gets paid. However, if Alice or Bob refuses to sign, then Donald can help resolving the transaction by signing. It is important to note that the semantic of the joint payment differs from the conditional transaction though: Bob cannot wait until he gets the money from Alice to choose what to do, whether to pay Carole.

6 Discussion

An interesting aspect of Nakamoto's consensus is that if the system is large enough and the mining power is sufficiently scattered among enough mining pools, then the probability of having a mining pool mining faster than the other can be made arbitrarily small. For this reason, the Blockchain anomaly has a very low chance of occurring in realistic executions of a large-scale permissionless blockchain systems like Bitcoin or Bitcoin-NG [9]. Recent work has shown, however, that incentives exist for miners to not disclose the block they successfully mine in order to waste the mining efforts of others, making it possible for them to mine a longer chain of blocks than others. This could potentially lead in turn to the Blockchain anomaly [10].

The 51-percent attack, where an attacker who controls more than half of the mining power of the public network can mine blocks faster than others, could lead to the Blockchain anomaly in a public blockchain system. The attacker can issue a transaction to convert some bitcoins to withdraw some money. Once the transaction is mined into a block at index i, then the attacker can fork the blockchain from index $i-1$, hence excluding his transaction, with a new series of blocks that gets eventually longer than the main chain. As the longest branch gets adopted, the attacker's transaction does not appear in the chain so that, in the end, the attacker withdrew some money while keeping his coins. With the same technique, one could easily override the block containing the transaction from Alice to Bob. The possibility of such an attack was raised in the context of the Bitcoin public chain as the mining power was noted as insufficiently scattered to avoid coalition [10].

One may think that the blockchain anomaly is specific to proof-of-work as there exist blockchain systems not based on proof-of-work that would not suffer from this issue because they trade availability for consistency (as discussed in Section 7). This is the case of some proof-of-stake blockchain systems, like Tendermint, that guarantees agreement and validity of consensus deterministically. The blockchain anomaly however applies even to blockchain systems based on proof-of-stake. For example, Casper is a proof-of-stake alternative to the GHOST reorganistion protocol used in Ehtereum. It looks like proof-of-stake does not necessarily solve the problem, as even Casper favours availability over consistency.[6]

Another problem raised by Gavin Wood, one of the founder of Ethereum, indicates that reorganisation can impact the initial order of transactions. This matters in an execution where two transactions aim at transfering $100 from the same account whose initial balance is only $100 because only the transaction that is committed first can be executed.[7] The Blockchain anomaly is more general than this problem, in particular the Blockchain anomaly allows conflicting transactions to be successfully executed and committed in two different states of the blockchain. Because the

[6] http://ethereum.stackexchange.com/questions/332/what-is-the-difference-between-casper-and-tendermint/536.

[7] https://blog.ethereum.org/2015/08/08/chain-reorganisation-depth-expectations/.

blockchain anomaly is more general, solving the blockchain anomaly would also solve this problem.

As it is known to be impossible to solve consensus in an asynchronous system in the presence of failures, researchers generally consider that a protocol ensures termination or agreement deterministically but not both. In this paper, we considered that the blockchain consensus terminates deterministically based on the recommended 6 to 12 mined blocks of Bitcoin [16] and Ethereum [21] but sometimes failing at ensuring agreement. Note that other formalisations also consider that termination of Nakamoto's consensus is deterministic and that only its safety property is probabilistic [12], just like we did. One may argue however that termination is not guaranteed deterministically but rather probabilistically and that one can increase the probability of consensus agreement by simply delaying the termination; the characterisation of Nakamoto's consensus in Bitcoin-NG adopts this definition [9]. In practice, however, blockchain applications assume consensus termination to provide a responsive service, as explained in Section 2.3. For example, Vitalik Buterin, one of the founder of Ethereum, explained that waiting for 12 mined blocks is probably sufficient for the first block to be irreversible.[8] This can be true in large-scale permissionless system where the mining power is sufficiently scattered among mining pools, but as the Blockchain anomaly shows, it is easy to revert it in a private chain context.

7 Related Work

Proof-of-work has been previously compared to Byzantine fault tolerant protocols [5, 19]. Some of this research [5] focuses on comparing experimentally Bitcoin against PBFT [4]. The Bitcoin blockchain and the PBFT consensus protocol were evaluated with nodes scattered at 8 locations around the world. As one could expect given the difficulty of the crypto puzzle of Bitcoin, the experiments showed that PBFT achieves a lower latency and a higher throughput than Bitcoin in serving transactions. However, PBFT suffers from scalability limitations and the authors recommend using sharding to avoid having to scale to hundreds of nodes.

Another part of this research [19] discusses the probabilistic guarantees of proof-of-work systems and the deterministic guarantees of Byzantine fault tolerance. The proof-of-work consensus is compared to Byzantine agreement protocols along two axes, scalability and performance, where proof-of-work consensus protocols are considered as scalable but inefficient while Byzantine agreement protocols are considered as efficient but not scalable. For example, Bitcoin scales beyond 1000 nodes while achieving a performance lower than 100 transactions per second with a high latency, whereas standard Byzantine fault tolerant protocols achieve more than 10,000 transactions per second but scale only to tens of nodes.

Some solutions immune to the Blockchain anomaly also exist. PeerCensus [6] was proposed as an algorithm with two components: one to execute a Byzantine agreement protocol on top of Bitcoin with a simple voting system and another to minimise the effect of Sybil attacks during these votes. The latter component makes it difficult for an attacker to create multiple identities so as to outnumber the votes with its own votes. Using this technique PeerCensus strengthens the guarantees of Bitcoin and resolves immediately the forks, hence avoiding the Blockchain anomaly.

Tendermint[9] is a blockchain system building upon proof-of-stake. It is known to favour consistency over availability, taking the opposite view of Casper, the proof-of-stake alternative to the GHOST protocol. The Tendermint consensus protocol builds upon the Byzantine agreement protocol with authentication [7] and requires strictly more than two third of correct processes to ensure agreement and validity deterministically and to guarantee termination when the network stabilises and messages between non-faulty nodes get delivered.

Although the Paxos anomaly was not considered a problem in the original design of Paxos [15], this scenario was informally stated as an anomaly during the design of the Zookeeper distributed coordination service [14], due to the engineers needing to implement conditional concurrent requests: Zookeeper organises nodes into a tree structure and it was desirable for the additions of a parent node and its child to be made concurrent. The child addition depended naturally on the success of the parent addition. Note that for other applications that do not need concurrent dependent requests Paxos is sufficient [13]. The Paxos anomaly differs from the Blockchain anomaly because it can occur on two transactions issued by the same client. In Blockchain systems, a timestamp can be used to order the transactions issued by the same client. Another major difference between the Paxos and the Blockchain anomalies is that if consensus is reached, the index of the decision cannot change while the Blockchain anomaly precisely stems from the fact that the index of a decided transaction, or the order of its block in the chain, can change.

8 Conclusion

This paper presents the Blockchain anomaly. Named after the Paxos anomaly, it prevents a user of mainstream blockchain systems from executing a conditional transaction, a transaction that should only execute in the current observable committed state or a later state of the system. Our experience of the use of an Ethereum private chain at NICTA/-Data61 revealed the easiness of reproducing the anomaly by reordering transactions after they had been committed. A possible way to avoid the anomaly could be to write smart contracts rather than transactions, yet it adds to the level of complexity.

Our conclusion is that blockchain systems are difficult to use properly. This observation should discourage users from using blockchain systems unless they fully understand the underlying design principles and the guarantees they offer.

[8] http://ethereum.stackexchange.com/questions/183/how-should-i-handle-blockchain-forks-in-my-dapp/203#203.

[9] http://tendermint.com.

If we combine the facts that blockchain applications require consensus to terminate fast while the underlying blockchain protocols guarantee agreement probabilistically then we can obtain dramatic results when applied to private chains.

Besides the prominent blockchain systems we have discussed, namely Bitcoin and Ethereum, there exist many alternatives. Exploring the alternatives that exclusively offer deterministic guarantees for private chain are part of future work.

Acknowledgements

We wish to thank our colleagues Alexander Ponomarev and Mark Staples for their feedback on an earlier version of this paper and for pointing out Gavin Wood's blog post on the insufficient fund problems of Ethereum. NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

References

[1] K. Birman, D. Malkhi, and R. van Renesse. Virtually synchronous methodology for dynamic service replication. Technical Report MSR-TR-2010-151, Microsoft Research, 2010.

[2] K. Birman, D. Malkhi, and R. van Renesse. *Guide to Reliable Distributed Systems: Building High-Assurance Applications and Cloud-Hosted Services*, chapter Virtually Synchronous Methodology for Building Dynamic Reliable Services, pages 635–671. Springer London, London, 2012.

[3] A. Black. Hashcash - a denial of service countermeasure. Technical report, Cypherspace, 2002.

[4] M. Castro and B. Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, Nov. 2002.

[5] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer, D. Song, and R. Wattenhofer. On scaling decentralized blockchains. In *3rd Workshop on Bitcoin Research (BITCOIN), Barbados*, February 2016.

[6] C. Decker, J. Seidel, and R. Wattenhofer. Bitcoin meets strong consistency. In *Proceedings of the 17th International Conference on Distributed Computing and Networking (ICDCN)*, page 13, 2016.

[7] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2), Apr. 1988.

[8] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '92, pages 139–147, 1993.

[9] I. Eyal, A. E. Gencer, E. G. Sirer, and R. van Renesse. Bitcoin-NG: A scalable blockchain protocol. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2016.

[10] I. Eyal and E. G. Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security - 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers*, pages 436–454, 2014.

[11] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, Apr. 1985.

[12] J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, pages 281–310, 2015.

[13] V. Gramoli, L. Bass, A. Fekete, and D. Sun. Rollup: Non-disruptive rolling upgrade with fast consensus-based dynamic reconfigurations. *IEEE Trans. on Parallel and Distributed Systems*, 2015.

[14] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed. Zookeeper: Wait-free coordination for internet-scale systems. In *ATC*, pages 11–11. USENIX, 2010.

[15] L. Lamport. The Part-Time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, May 1998.

[16] S. Nakamoto. Bitcoin: a peer-to-peer electronic cash system, 2008. http://www.bitcoin.org.

[17] D. Ongaro and J. Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 305–319, Philadelphia, PA, 2014. USENIX Association.

[18] N. Szabo. Formalizing and securing relationships on public networks, 1997. http://szabo.best.vwh.net/formalize.html.

[19] M. Vukolíc. The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication. In *Proceedings of the IFIP WG 11.4 Workshop on Open Research Problems in Network Security (iNetSec 2015)*, LNCS, 2016.

[20] G. Wood. Ethereum: A secure decentralised generalised transaction ledger final draft - under review, 2014. https://github.com/ethereum/wiki/wiki/White-Paper.

[21] G. Wood. Ethereum: A secure decentralised generalised transaction ledger, 2015. Yellow paper.

[22] X. Xu, C. Pautasso, L. Zhu, V. Gramoli, S. Chen, A. Ponomarev, and A. B. Tran. The blockchain as a software connector. In *Proceedings of the 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, April 2016.