The Blockchain Anomaly

Christopher Natoli
NICTA/Data61-CSIRO
University of Sydney
chistopher.natoli@sydney.edu.au

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

***Abstract*—Most popular blockchain solutions rely on proof-of-work to guarantee that participants reach consensus on a unique block per index of the chain. As consensus is impossible in the general case, it seems that these blockchain systems require messages are delivered fast and no participant mines faster than the crowd. To date, no experimental settings have however been proposed to demonstrate this hypothesis.**

In this paper, we identify conditions under which these blockchain systems fail to ensure consensus and present a reproducible execution on our Ethereum private chain. To this end, we introduce the *Blockchain Anomaly*, the impossibility for the blockchain to guarantee that a committed transaction is not abortable. This anomaly may translate into dramatic consequences for the user of proof-of-work 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 and may lead to double spending. We also explain how the anomaly differs from a 51-percent attack and how one could avoid it by adapting the Ethereum implementation or by exploiting smart contracts.

Keywords: Ethereum; Paxos anomaly; smart contract

I. Introduction

Mainstream public blockchain systems, like Bitcoin [1] and Ethereum [2], require to reach consensus on the Internet despite the presence of malicious participants and possible congestions that delay messages. Yet, it is impossible for a distributed system including a faulty process to reach consensus if messages may not be delivered within a bounded time [3]. This raises interesting questions about the properties ensured by blockchains. Foundational consensus algorithms [4] 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, especially when delays occur [5]. 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. The R3 consortium is currently running an Ethereum private chain with more than 45 banks worlwide.[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 [6], [7], [8], 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. These results confirm the risk of using a blockchain in a private context without understanding its complex design features, which also confirms the need for solid research foundations [9]. We terminate this paper 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 its branches, the "longest" (i.e., heaviest in Ethereum or deepest in Bitcoin) 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 cryptopuzzle that boils down to computing for 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 [2].

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 [3]. Consensus is usually expressed in 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 [10] and Raft [11], is to make sure that if the messages get delayed, at least

[1] http://www.coindesk.com/r3-ethereum-report-banks.

978-1-5090-3216-7/16/$31.00 ©2016 IEEE

validity and agreement remain ensured by having the algorithm doing nothing, hence sacrificing termination to ensure that only correct responses—satisfying both validity and agreement—can be returned. These "indulgent" consensus algorithms [12] are appealing, because if after some time the network stabilizes and messages get delivered in a bounded time, then consensus can be reached [4].

We show experimentally that the Ethereum protocol can suffer from the Blockchain anomaly. We 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. Although there exist solutions in the distributed computing literature to order some transactions [13] or to use unstructured overlays to cope with malicious participants [14], to our knowledge no experiment has ever been proposed to show that proof-of-work protocols are subject to such a problem.

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.

Section II overviews the blockchain technology, the Paxos anomaly and defines the important terms of the paper. In Section III, we present the blockchain anomaly. In Section IV, we present our experiments and illustrate how the anomaly is possible with less than half of the mining power. In Section V, we explain how replacing transactions by smart contracts could help bypassing the anomaly. Section VI presents the related work. And Section VII concludes.

II. 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. We consider a distributed blockchain system of n peers where peers can exchange coins from one to another through *transactions*. 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 special peers, called *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.

A. Blockchain Systems

A blockchain can be considered as a replicated state machine [15] where a reversed link between blocks is a pointer from a state to its preceding state as depicted in Figure 1(a). 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 [16]. Miners provably solve a hashcash crypto puzzle [17] to append a new block to the chain. 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.

B. From Nakamoto's Consensus to Smart Contracts

Nakamoto's consensus [1] 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).

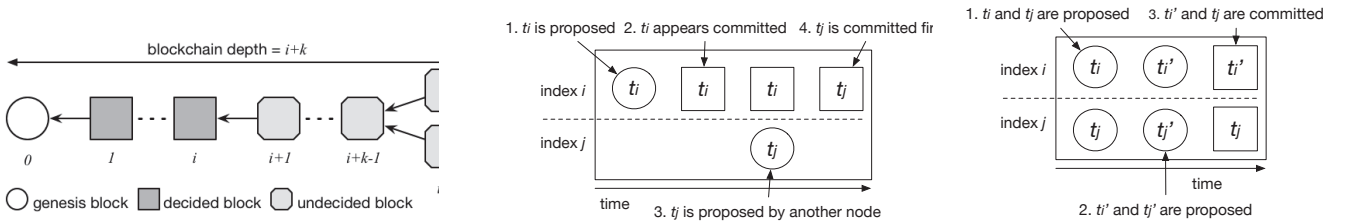
Ethereum [18] is a recent open source cryptocurrency platform that also builds upon proof-of-work. As opposed to Bitcoin's consensus protocol, Ethereum generates one block every 12–15 seconds. While it improves the throughput (transactions per second) it also favors transient forks as miners are more 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 discard 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* [19] that define new ownership rules.

C. 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 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*

[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.



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

(b) The Blockchain anomaly: a first client issues that gets successfully mined and committed the second client issues t_j, with t_j being conditio to the commit of t_i (note that $j \geq i+k$ t_i to be committed before t_j gets issued), the transaction t_j gets finally reorganized and s cessfully committed before t_i, hence violating dependency between t_i and t_j

(c) 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

Fig. 1: An example of decided blocks and the difference between the Paxos and the Blockchain anomaly

(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(a) depicts the termination of consensus on the index i of a blockchain starting with the *genesis* block. An arrow pointing from right to left 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.

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 version 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 $(k_{eth}+1)*15\,sec = 3\,min$ 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]

D. The Paxos Anomaly

Paxos is a famous consensus protocol originally guaranteeing agreement and validity despite crash failures [10]. The *Paxos anomaly* [7], [6] 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 1(c) 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 [6]. However, in Paxos, as in fully decentralised systems, the first transaction may not be decided in favor 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.

III. The Blockchain Anomaly

We present the Blockchain anomaly, an anomaly of blockchain consensus protocols.

A. 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 network where users

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

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 (consortium or fully) private chain where the employees of an institution, like Data61-CSIRO, have direct access to some of the network resources. When messages get finally delivered, the results of the disagreement creates inconsistencies.

B. Uncommitting Transactions is Abnormal

Figure 1(b) depicts the Blockchain anomaly, where a transaction t_i gets committed as part of slot i. After observing that t_i is committed, a node 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.

C. 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 is similar to a double-spending attack against Bitcoin [20] and 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 which it can collect its goods as a result of transaction t_1, then it publicizes 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.

D. 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 incentivizes participants to leverage the Blockchain anomaly to attack the private chain.

IV. Experimental Evaluation

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

A. 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. The same setting as the one used above could allow us to conduct a 51-percent attack, however, the 51-percent attack is not necessary to encounter a blockchain anomaly. For example in a private blockchain adopting the longest branch, if the attacker only owns a minority of the mining power then not adapting the block size or the difficulty of the crypto-puzzle adequately with respect to the network delay could result in having the system adopting p_3's branch anyway.

B. Distributed Execution

For the sake of reproducibility, we present a simple execution where a malicious miner mines faster than a correct miner, the discussion of the anomaly when the malicious miner owns less than half of the mining power is defer to Section IV-E. 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 2 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'.

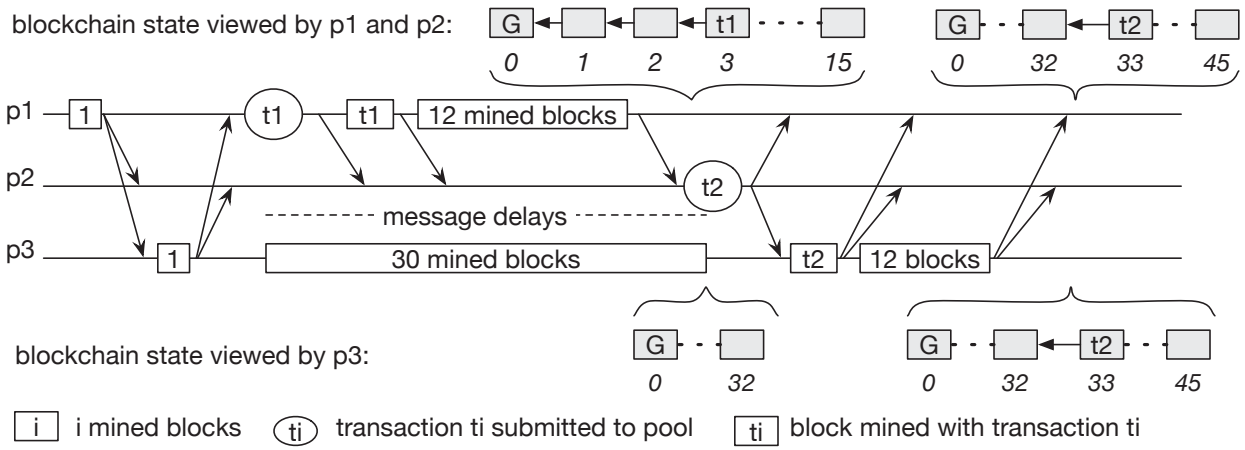


Fig. 2: Execution scenario leading to the Blockchain anomaly: p_3 mines a longer chain than p_1 without including t_1 and without disseminating new blocks until it forces a reorganisation that imposes t_2 to be committed while t_1 appears finally uncommitted

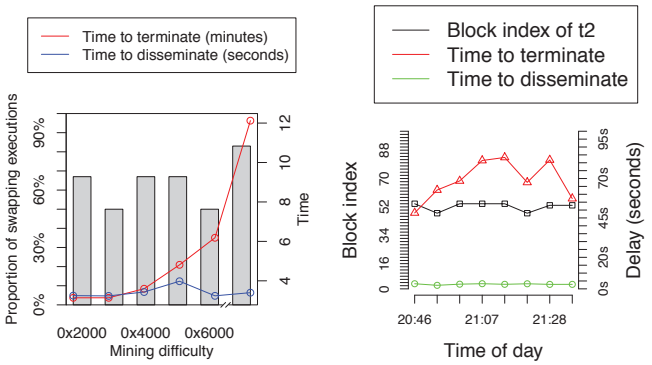
1) Peer p_1 mines a first block after the genesis block and informs p_2 and p_3 to update their view of the blockchain state.
2) Peer p_3 mines a second block and informs p_1 and p_2 of this new block.
3) A network delay is introduced between peers p_1 and p_2 on the one hand, and peer p_3 on the other 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.

C. Automating the Reproduction of the Anomaly

To illustrate the anomaly, we wrote a script that automated the execution depicted in Figure 2. Figure 3(b) represents the execution of a script that execute 8 iterations of the Blockchain anomaly over a period of 50 minutes. Again the 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 1(b) 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.



(a) The proportion of transaction swaps observed does not depend on the difficulty, as opposed to the consensus termination that increases with the difficulty

(b) Automated executions of the Blockchain anomalies over a period of $50\,min$, the execution is non-deterministic due to the randomness of the mining process and the network delay between peers

Fig. 3: Experimental evaluation of the anomaly

This is due to the time needed to mine a block, which is significantly larger than the latency of our network.

D. Swap Frequency with Different Mining Difficulties

In the previous experiment, we used the default Ethereum difficulty (0x4000) 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 3(a) 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 2.

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 IV-C.)

We varied the difficulty from 0x2000 to 0x40000 and measure the frequency of the Blockchain anomaly and the time it would take for consensus to terminate (upper curve). We observed that the termination time was proportional to the difficulty while the occurrence of the anomaly was not significantly affected by the difficulty. This is explained by the fact that the difficulty impacts the time it takes to mine $k+1$ blocks for termination. 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.

E. The Blockchain Anomaly Differs from the 51-Percent Attack

For the sake of reproducibility we simplified the execution leading to the Blockchain anomaly in Figure 2. It is however important to note that the Blockchain anomaly can occur even though the malicious user controls less than half of the mining power. To this end, Figure 4 depicts an execution where peer

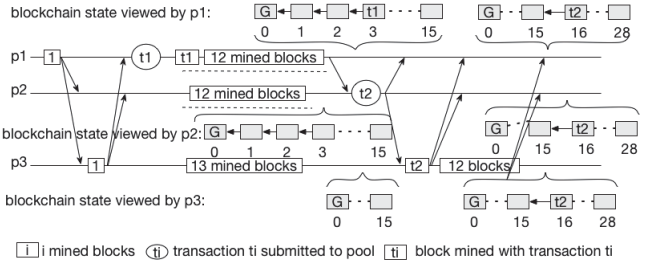


Fig. 4: Half of the mining power is not necessary for the blockchain anomaly as it is sufficient to discard the blockchain containing t_1

```
contract conditionalPayment {
  // to keep track of the amount paid by Alice
  uint32 paid;
  // map addresses to their respective balance
  mapping (address => uint256) public balances;
  // the address of Alice's account
  address A = 0x57ec7927841e2d25aad5f335e3b701369b177392;
  // the address of Bob's account
  address B = 0x5ae58375c89896b09045de349289af9034902905;
  // the address of Carole's account
  address C = 0x3b12387c88de7834ab3129e3949d0918c4a09122;

  // enables function depending on invoker
  modifier onlyFrom(address _address) {
    if (msg.sender != _address) throw;
    _
  }

  // Alice sends money to Bob
  function sendTo(address B, uint32 _amount) onlyFrom(A) {
    if (balances[A] >= _amount) { // sufficient funds?
      balances[A] -= _amount;
      balances[B] += _amount;
      paid = _amount; // sorting the amount paid
    }
  }

  // Bob sends money to Carole
  function sendIfReceived(address C, uint32 _amount) onlyFrom(B) {
    if (paid > _amount) { // only if previous payment
      balances[B] -= _amount;
      balances[C] += _amount;
    } else {
      throw; // cancel contract execution
    }
  }
}
```

Fig. 5: 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

p_3 owns strictly less than half of the mining power. As the network is delayed between all pairs of nodes, we can see that p_1 alone cannot mine a chain longer than p_3's and that the blockchain of p_1 containing t_1 gets eventually overridden.

V. 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

```
1  contract problematicConditionalPayment {
2    ...
3    function checkPayment(address B, uint32 _amount) onlyFrom(B)
         constant returns (bool result) {
4      if (paid > _amount) { // check that Alice paid
5        return true;
6      } else throw;
7    }
8    // Bob sends money to Carole
9    function sendIfReceived(address C, uint32 _amount) onlyFrom(B) {
10     balances[B] -= _amount;
11     balances[C] += _amount;
12   }
13 }
```

Fig. 6: Executing the transfer to Carole in a separate function may suffer from the Blockchain anomaly

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.

In Figure 5, 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 6 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 reorganized while Bob was checking the result off-line.

To conclude, the former contract in Figure 5 does not suffer from the Blockchain anomaly as it executes the check and the conditional transfer on-chain.

VI. Related work

Proof-of-work has been previously compared to Byzantine Fault Tolerant protocols [21], [22]. Some of this research [21] focuses on comparing experimentally Bitcoin against PBFT [23]. 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 using sharding [24] could be necessary to scale to hundreds of nodes.

Another part of this research [22] 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.

Multiple attacks to Bitcoin share the "solo-mining" technique we used to illustrate the Blockchain anomaly in Ethereum. Some attacks assume that the merchant accepts transactions before they are confirmed [25], [26], [27]. Other attacks assume the merchant to accept transactions that are confirmed once [28]. According to our definition none of these transactions are however "committed", hence these attacks cannot be considered anomalies. The Blockchain anomaly affects transactions that are committed (or $k+1$ times confirmed).

Even though more than half of the mining power was controlled by the adversary to illustrate a simple scenario to reproduce the Blockchain anomaly, it is important to notice that the Blockchain anomaly is different from the 51-percent attack. First, note that the smart contract solution we proposed in Section V cannot fix the 51-percent attack. More generally, the blockchain anomaly could occur with n nodes with all attackers owning totally a q-th of the mining power if each correct node mines at a rate of q/n blocks every block propagation delay.

Some solutions to the Blockchain anomaly could be easily implemented in Ethereum. As an example, logical clocks is a well-known technique to order causally-related events in a distributed system [13]. A logical clock could be used to order two transactions issued by the same peer, simply associating messages to sequence numbers using a monotonically increasing counter at each peer. As this cannot be used for dependent transactions issued by different peers, one could use a special flag when issuing a transaction to inform the miners to either ignore the transaction or to mine it within the same block as its precedent dependent transaction. Designing the reward model to incentivize the miners to follow this protocol is out of the scope of this paper. Other technique to perform a transaction as soon as other were performed were used to enhance the scalability of Bitcoin [29].

Some solutions immune to the Blockchain anomaly also exist. PeerCensus [30] 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 minimize 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.

Although the Paxos anomaly was not considered a problem in the original design of Paxos [10], this scenario was informally stated as an anomaly during the design of the Zookeeper distributed coordination service [6], due to the engineers needing to implement conditional concurrent requests: Zookeeper organizes 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 [31]. A major difference between the Paxos and the Blockchain anomalies is that if consensus is reached with Paxos, 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.

VII. Conclusion

In this paper, we demonstrate empirically the presence of the Blockchain anomaly in proof-of-work blockchain systems. 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. 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. 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 chains is part of future work.

References

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

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

[3] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *J. ACM*, vol. 32, no. 2, pp. 374–382, Apr. 1985.

[4] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *J. ACM*, vol. 35, no. 2, Apr. 1988.

[5] Y. Sompolinsky and A. Zohar, "Secure high-rate transaction processing in bitcoin," in *Financial Cryptography and Data Security - 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers*, 2015, pp. 507–527.

[7] K. Birman, D. Malkhi, and R. van Renesse, "Virtually synchronous methodology for dynamic service replication," Microsoft Research, Tech. Rep. MSR-TR-2010-151, 2010.

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

[8] ——, "Appendix A: Virtually synchronous methodology for building dynamic reliable services," in *Guide to Reliable Distributed Systems: Building High-Assurance Applications and Cloud-Hosted Services*. London: Springer London, 2012, pp. 635–671.

[9] V. Gramoli, "On the danger of private blockchains," in *Workshop on Distributed Cryptocurrencies and Consensus Ledgers (DCCL)*, July 2016.

[10] L. Lamport, "The Part-Time parliament," *ACM Transactions on Computer Systems*, vol. 16, no. 2, pp. 133–169, May 1998.

[11] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. Philadelphia, PA: USENIX Association, 2014, pp. 305–319. [Online]. Available: https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro

[12] R. Guerraoui, "Indulgent algorithms (preliminary version)," in *PODC*, 2000, pp. 289–297.

[13] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, Jul. 1978.

[14] E. Anceaume, Y. Busnel, and S. Gambs, "On the power of the adversary to solve the node sampling problem," *Trans. Large-Scale Data- and Knowledge-Centered Systems*, vol. 11, pp. 102–126, 2013.

[15] 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)*, 2016.

[16] 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*, ser. CRYPTO '92, 1993, pp. 139–147.

[17] A. Black, "Hashcash - a denial of service counter-measure," Cypherspace, Tech. Rep., 2002. [Online]. Available: http://www.hashcash.org/papers/hashcash.pdf

[18] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," 2015, yellow paper.

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

[20] M. Rosenfeld, "Analysis of hashrate-based double-spending," 2012.

[21] 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.

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

[23] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Trans. Comput. Syst.*, vol. 20, no. 4, pp. 398–461, Nov. 2002.

[24] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proceedings of the 23rd ACM Conference on Computer and Communications Security (CCS)*, 2016.

[25] H. Finney, "Finney's attack," February 2011. [Online]. Available: https://bitcointalk.org/index.php?topic=3441.msg48384#msg48384

[26] G. Karame, E. Androulaki, and S. Capkun, "Two bitcoins at the price of one? double-spending attacks on fast payments in bitcoin," *IACR Cryptology ePrint Archive*, vol. 2012, p. 248, 2012.

[27] T. Bamert, C. Decker, L. Elsen, R. Wattenhofer, and S. Welten, "Have a snack, pay with bitcoins," in *Proceedings of the 13th IEEE International Conference on Peer-to-Peer Computing (P2P)*, 2013, pp. 1–5.

[28] vector76, "The vector76 attack," August 2011. [Online]. Available: https://bitcointalk.org/index.php?topic=36788.msg463391#msg463391

[29] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," 2016, draft Version 0.5.9.2. [Online]. Available: https://lightning.network/lightning-network-paper.pdf

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

[31] 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*, 2016.