The Balance Attack or Why Forkable Blockchains Are Ill-Suited for Consortium

Christopher Natoli
School of IT
University of Sydney
Sydney, Australia
christopher.natoli@sydney.edu.au

Vincent Gramoli
School of IT
Data61-CSIRO and University of Sydney
Sydney, Australia
vincent.gramoli@sydney.edu.au

***Abstract*—Most blockchain systems are *forkable* in that they require participants to agree on a chain out of multiple possible branches of blocks. In this paper, we identify a new form of attack, called the Balance attack, against these forkable blockchain systems. The novelty of this attack consists of delaying network communications between multiple subgroups of nodes with balanced mining power. Our theoretical analysis captures the tradeoff between the network delay and the mining power of the attacker needed to double-spend in the GHOST protocol with high probability.**

We quantify our analysis in the settings of the Ethereum testnet of the R3 consortium where we show that a single machine needs to delay messages for 20 minutes to double spend while a coalition with a third of the mining power would simply need 4 minutes to double spend with 94% of success. We experiment the attack in our private Ethereum chain before arguing for a non-forkable blockchain design to protect against Balance attacks.

***Keywords*-Ethereum, forks, GHOST, consortium blockchain, private blockchain, Bitcoin, Byzantine, Casper**

I. Introduction

Blockchain systems are distributed implementations of a chain of blocks. Each node can issue a cryptographically signed transaction to transfer digital assets to another node or can create a new block of transactions, and append this block to its current view of the chain. Due to the distributed nature of this task, multiple nodes may append distinct blocks at the same index of the chain before learning about the presence of other blocks, hence leading to a forked chain or a *tree*. For nodes to eventually agree on a unique state of the system, nodes apply a common strategy that selects a unique branch of blocks in this tree.

Bitcoin [33], one of the most popular blockchain systems, selects the longest branch. This strategy has however shown its limitation as it simply *wastes all blocks* not present in this branch [11], [37], [39], [20], [36]. If an attacker can solve crypto-puzzles fast enough to grow a local branch of the blockchain faster than the rest of the system, then it will eventually impose its own branch to all participants. In particular, by delaying the propagation of blocks in the system, one can increase the amount of wasted blocks and proportionally slow down the growth of the longest branch of the system. This delay presents a serious risk to the integrity of the blockchain, as the attacker does not even need a large fraction of the computational power to exceed the length of the chain, allowing her to *double spend* in new transactions the coins that she already spent in earlier transactions [38].

Ethereum [41] proposes another selection strategy that copes with this problem. Each node uses an algorithm, called GHOST, that starts from the first block, also called the *genesis block*, and iteratively selects the root of the heaviest subtree to construct the common branch. Even if nodes create many blocks at the same index of the blockchain, their computational power is not wasted but counted in the selection strategy [39]. In particular, the number of these "sibling" blocks increase the chance that their common ancestor block be selected in favor of another candidate block mined by the attacker. Although it clearly alleviates the Bitcoin limitation discussed above [11], [37], [20], [36] it remains unclear how long an attacker with a low mining power should delay messages to discard previous transactions in Ethereum.

In this paper, we answer this question by demonstrating theoretically and experimentally that an attacker can compensate a low mining power by delaying selected messages in Ethereum. To this end, we propose a simple attack, called the *Balance attack*: an attacker transiently disrupts communications between subgroups of similar mining power. During this time, the attacker issues transactions in one subgroup, say the *transaction subgroup*, and mines blocks in another subgroup, say the *block subgroup*, up to the point where the tree of the block subgroup outweighs, with high probability, the tree of the transaction subgroup. The novelty of the Balance attack is to leverage the GHOST protocol that accounts for sibling or *uncle* blocks to select a chain of blocks. This strategy allows the attacker to mine a branch possibly in isolation of the rest of the network before merging its branch to one of the competing blockchain to influence the branch selection process.

We experimented a distributed system running Ethereum in similar settings as R3, a consortium of more than 70 world-wide financial institutions. In January, R3 consisted of eleven banks and successfully collaborated in deploying an

Ethereum private chain to perform transactions.[1] Since then, R3 has grown and kept using Ethereum[2] while the concept of *consortium private chain* gained traction for its ability to offer a blockchain system among multiple companies in a not necessarily private but controlled environment. R3 has experimented with various blockchains and has recently released a distributed ledger implementation that does not use any blockchain [5]. As opposed to a fully private chain scenario, the consortium private chain involves different institutions, possibly competitors. As they can be located at different places around the world, they typically use Internet to communicate. We illustrate the Balance attack in the R3 network setting as of June 2016 that consisted of a testnet of 50 machines among which 15 were actively mining, by deploying Ethereum on own private testnet.

One can exploit the Balance attack to *corrupt* or violate the persistence of the main branch, hence rewriting previously committed transactions, and allowing the attacker to double spend. As opposed to previous attacks against Bitcoin where the attacker has to expand the longest chain faster than correct miners to obtain this result [38], the novelty of our attack lies in the contribution of the attacker to one of the correct miner chain in order to outweigh another correct miner chain. We generalize our contribution to *forkable* blockchains, i.e., blockchains that experience forks, by proposing a simple model for forkable blockchains and specifying Nakamoto's and GHOST consensus algorithmic differences. More precisely, we make the four following contributions:

1) We show that the GHOST consensus protocol is *corruptible* or vulnerable to double spending attacks with high probability when an attacker can delay communications. We introduce the Balance attack that influences the branch selection rather than trying to solo-mine heavier or longer branches.
2) We illustrate the problem in the context of the R3 Ethereum testnet if a single node can delay communications for 20 minutes or when a coalition delay communications during 4 minutes. This vulnerability stems from the heterogeneous nature of participants that is exacerbated in small consortium blockchains.
3) We argue for a non-forkable blockchain design that protects against Balance attacks. The idea is to design fork-free blockchains so that no adversary can exploit forks to influence the decision in favor of a particular branch. This design may find applications in critical sectors where the loss of digital assets cannot be tolerated.

As of today, mainstream blockchains have implicitly assumed *synchrony* [17] to be able to predict an upper-bound on the delay of messages. While strong guarantees can clearly be proved under this assumption, the guarantees to expect if this assumption is violated remain unclear. The problem is that mainstream blockchains, whether they are fully public or involve a consortium of institutions, relies on large network, like the internet, in which one cannot predict the delay of messages. More dramatically, multiple results demonstrated recently how one can simply delay messages in blockchain networks [25], [22], [1]. The Balance attack reveals that delaying messages can be dramatic for blockchains, leading to double-spending even with a small portion of the mining power and raises interesting challenges in the design of safe consortium blockchains.

Section II defines the problem. In Section III, we present the algorithm to run the attack. In Section IV, we show how the analysis affects GHOST. In Section V, we analyze the success of the Balance attack in the context of the R3 consortium network. In Section VI, we present our experiments run in an Ethereum private chain. In Section VII, we propose some blockchain design solutions immune to the Balance attack. Section VIII presents the related work. And Section IX concludes.

II. PRELIMINARIES

In this section we model a simple distributed system that implements a blockchain abstraction as a directed acyclic graph. We propose a high-level pseudocode representation of proof-of-work blockchain protocols in this model that allows us to illustrate an important difference between Bitcoin and Ethereum in the selection of a main branch with a persistent prefix.

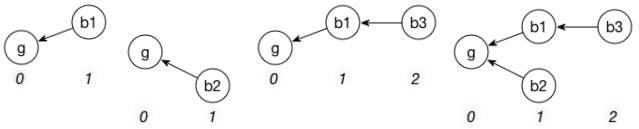
A. A simple distributed model for blockchains

We consider a communication graph $G = \langle V, E \rangle$ with nodes V connected to each other through fixed communication links E. Nodes are part of a blockchain system $S \in \{\mathsf{bitcoin}, \mathsf{ethereum}\}$ and can act as clients by issuing transactions to the system and/or servers by *mining*, the action of trying to combine transactions into a block. For the sake of simplicity, we consider that each node possesses a single account and that a *transaction* issued by node p_i is a transfer of digital assets or *coins* from the account of the source node p_i to the account of a destination node $p_j \neq p_i$. Each transaction is uniquely identified and broadcast to all nodes in a best-effort manner. We assume that a node re-issuing the same transfer multiple times creates as many distinct transactions.

Nodes that mine are called *miners*. We refer to the computational power of a miner as its *mining power* and we denote the total mining power t as the sum of the mining powers of all miners in V. Each miner tries to group a set T of transactions it heard about into a block $b \supseteq T$ as long as transactions of T do not conflict and that the account balances remain non-negative. For the sake of simplicity in

[1] http://www.ibtimes.co.uk/r3-connects-11-banks-distributed-ledger-using-ethereum-microsoft-azure-1539044.

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



(a) view ℓ_1 (b) view ℓ_2 (c) view ℓ_3 (d) global state $\ell_0 = \ell_1 \cup \ell_2 \cup \ell_3$

Figure 1: The global state ℓ_0 of a blockchain results from the union of the distributed local views ℓ_1, ℓ_2 and ℓ_3 of the blockchain

the presentation, the graph G is static meaning that no nodes can join and leave the system, however, nodes may fail as described in Section II-A2. As we consider a network potentially shared by other applications and subject to contention, we assume that the system is *partially synchronous* in that there is a bound on the delay of messages but that this bound can be large and is not known by the algorithm [12].

1) Miners try to create new blocks: Miners have the role of creating blocks, by sometimes provably solving a hashcash crypto-puzzle [3]. Given a global threshold and the block of largest index the miner knows, trying to solve a crypto-puzzle consists of repeatedly selecting a nonce and applying a pseudo-random function to this block and the selected nonce until a result lower than the threshold is obtained. Upon success the miner creates a block that contains the successful nonce as a proof-of-work as well as the hash of the previous block, hence fixing the index of the block, and broadcasts the block. As there is no known strategy to solve the crypto-puzzle, the miners simply keep testing whether randomly chosen numbers solve the crypto-puzzle. The mining power is thus expressed in the number of hashes the miner can test per second, or H/s for short. The *difficulty* of this crypto-puzzle, defined by the threshold, limits the rate at which new blocks can be generated by the network.

2) The failure model: We assume the presence of an *adversary* that can control *attacker* or malicious nodes that together own a relatively small fraction $0 < \rho < \frac{1}{2}$ of the total mining power of the system. The nodes controlled by the adversary may not follow the protocol specification, however, they cannot impersonate other nodes while issuing transactions.[3] A node that is not malicious is *correct*.

3) The forkable blockchain abstraction: Let the *blockchain* be a directed acyclic graph (DAG) $\ell = \langle B, P \rangle$ such that blocks of B point to each other with pointers P (pointers are recorded in a block as a hash of the previous block) and a special block $g \in B$, called the *genesis block*, does not point to any block but serves as the common first block.

[3]This is typically ensured through public key crypto-systems.

Algorithm 1 Blockchain construction at node p_i

1: $\ell_i = \langle B_i, P_i \rangle$, the local blockchain at node p_i is a directed acyclic
2: graph of blocks B_i and pointers P_i

3: receive-blocks$(\langle B_j, P_j \rangle)_i$: ▷ *upon reception of blocks*
4: $B_i \leftarrow B_i \cup B_j$ ▷ *update vertices of blockchain*
5: $P_i \leftarrow P_i \cup P_j$ ▷ *update edges of blockchain*

Algorithm 1 describes the progressive construction of a forkable blockchain at a particular node p_i upon reception of blocks from other nodes by simply aggregating the newly received blocks to the known blocks (lines 3–5). As every added block contains a hash to a previous block that eventually leads back to the genesis block, each block is associated with a fixed index. By convention we consider the genesis block at index 0, and the blocks at j hops away from the genesis block as the blocks at index j. As an example, consider the simple blockchain $\ell_1 = \langle B_1, P_1 \rangle$ depicted in Figure 1(a) where $B_1 = \{g, b_1\}$ and $P_1 = \{\langle b_1, g \rangle\}$. The genesis block g has index 0 and the block b_1 has index 1.

4) Forks as disagreements on the blocks at a given index: As depicted by views ℓ_1, ℓ_2 and ℓ_3 in Figures 1(a), 1(b) and 1(c), respectively, nodes may have a different views of the current state of the blockchain. In particular, it is possible for two miners p_1 and p_2 to mine almost simultaneously two different blocks, say b_1 and b_2. If neither block b_1 nor b_2 was propagated early enough to nodes p_2 and p_1, respectively, then both blocks would point to the same previous block g as depicted in Figures 1(a) and 1(b). Because network delays are not predictable, a third node p_3 may receive the block b_1 and mine a new block without hearing about b_2. The three nodes p_1, p_2 and p_3 thus end up having three different local views of the same blockchain, denoted $\ell_1 = \langle B_1, P_1 \rangle$, $\ell_2 = \langle B_2, P_2 \rangle$ and $\ell_3 = \langle B_3, P_3 \rangle$.

We refer to the *global blockchain* as the directed acyclic graph $\ell_0 = \langle B_0, P_0 \rangle$ representing the union of these local blockchain views, denoted by $\ell_1 \cup \ell_2 \cup \ell_3$ for short, as depicted in Figure 1, and more formally defined as follows:

$$\begin{cases} B_0 &= \cup_{\forall i} B_i, \\ P_0 &= \cup_{\forall i} P_i. \end{cases}$$

The point where distinct blocks of the global blockchain DAG have the same predecessor block is called a *fork*. As an example Figure 1(d) depicts a fork with two branches pointing to the same block: g in this example.

In the remainder of this paper, we refer to the DAG as a *tree* rooted in g with upward pointers, where children blocks point to their parent block.

5) Main branch in Bitcoin and Ethereum: To resolve the forks and define a deterministic state agreed upon by all nodes, a blockchain system must select a *main branch*, as a unique sequence of blocks, based on the tree. Building upon the generic construction (Alg. 1), we present two

selections: Nakamoto's consensus protocol (Alg. 2) present in Bitcoin [33] and the GHOST consensus protocol (Alg. 3) present in Ethereum [41].

Nakamoto's consensus algorithm: The difficulty of the crypto-puzzles used in Bitcoin produces a block every 10 minutes in expectation. The advantage of this long period, is that it is relatively rare for the blockchain to fork because blocks are rarely mined during the time others are propagated to the rest of the nodes.

Algorithm 2 Nakamoto's consensus protocol at node p_i

```
6:  m = 5, the number of blocks to be appended after the one containing
7:      tx, for tx to be committed in Bitcoin

8:  get-main-branch()_i:                          ▷ select the longest branch
9:      b ← genesis-block(B_i)                    ▷ start from the blockchain root
10:     while b.next ≠ ⊥ do                       ▷ prune shortest branches
11:         block ← argmax_{c∈children(b)}{depth(c)}   ▷ deepest subtree
12:         B ← B ∪ {block}                       ▷ update vertices of main branch
13:         P ← P ∪ {⟨block, b⟩}                  ▷ update edges of main branch
14:         b ← block                             ▷ move to next block
15:     return ⟨B, P⟩                             ▷ returning the Bitcoin main branch

16: depth(b)_i:                                   ▷ depth of tree rooted in b
17:     if children(b) = ∅ then return 1          ▷ stop at leaves
18:     else return 1 + max_{c∈children(b)} depth(c)   ▷ recurse at children
```

Algorithm 2 depicts the Bitcoin-specific pseudocode that includes Nakamoto's consensus protocol to decide on a particular block at index i (lines 8–18) and the choice of parameter m (line 6) explained later in Section II-B. When a fork occurs, Nakamoto's protocol resolves it by selecting the longest branch as the main branch (lines 8–15) by iteratively selecting the root of the deepest subtree (line 11). When process p_i is done with this pruning, the resulting branch becomes the main branch $\langle B_i, P_i \rangle$ as observed by the local process p_i. Note that the pseudocode for checking whether a block is decided and a transaction committed based on this parameter m is common to Bitcoin and Ethereum, it is thus deferred to Alg. 4.

6) The GHOST *consensus algorithm:* As opposed to the Bitcoin 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 without having heard about the latest mined blocks yet. To avoid wasting large mining efforts while resolving forks, Ethereum uses the GHOST (Greedy Heaviest Observed Subtree) consensus algorithm that accounts for the, so called *uncles*, blocks of discarded branches. In contrast with Nakamoto's protocol, the GHOST protocol iteratively selects, as the successor block, the root of the subtree that contains the largest number of nodes (cf. Algorithm 3).

The main difference between Nakamoto's consensus protocol and GHOST is depicted in Figure 2, where the black blocks represent the main branch selected by Nakamoto's consensus protocol and the grey blocks represent the main branch selected by GHOST. Note that Ethereum adapted GHOST to account for the difficulties of blocks in the choice of the heaviest subtree [39].

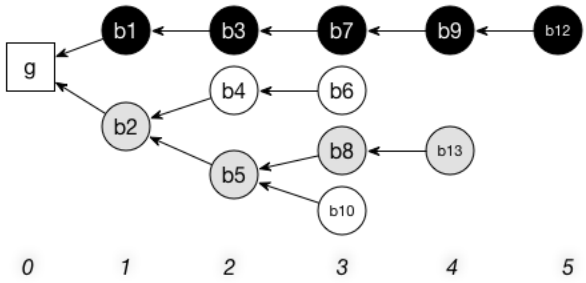


Figure 2: Nakamoto's consensus protocol at the heart of Bitcoin selects the main branch as the longest branch (in black) whereas the GHOST consensus protocol at the heart of Ethereum follows the heaviest subtree (in grey)

Algorithm 3 The GHOST consensus protocol at node p_i

```
6:  m = 11, the number of blocks to be appended after the one containing
7:      tx, for tx to be committed in Ethereum (since Homestead v1.3.5)

8:  get-main-branch()_i:                          ▷ select the branch with the most nodes
9:      b ← genesis-block(B_i)                    ▷ start from the blockchain root
10:     while b.next ≠ ⊥ do                       ▷ prune lightest branches
11:         block ← argmax_{c∈children(b)}{num-desc(c)}   ▷ heaviest tree
12:         B ← B ∪ {block}                       ▷ update vertices of main branch
13:         P ← P ∪ {⟨block, b⟩}                  ▷ update edges of main branch
14:         b ← block                             ▷ move to next block
15:     return ⟨B, P⟩.                            ▷ returning the Ethereum main branch

16: num-desc(b)_i:                                ▷ number of nodes in tree rooted in b
17:     if children(b) = ∅ then return 1          ▷ stop at leaves
18:     else return 1 + Σ_{c∈children(b)} num-desc(c)   ▷ recurse at children
```

B. Decided blocks and committed transactions

A blockchain system S must define when the block at an index is agreed upon. To this end, it has to define a point in its execution where a prefix of the main branch can be "reasonably" considered as persistent.[4] More precisely, there must exist a parameter m provided by S for an application to consider a block as *decided* and its transactions as *committed*. This parameter is typically $m_{bitcoin} = 5$ in Bitcoin (Alg. 2, line 6) and $m_{ethereum} = 11$ in Ethereum (Alg. 3, line 6). Note that these two choices do not lead to the same probability of success [18] and different numbers are used by different applications [35].

Definition 1 (Transaction commit)**.** *Let $\ell_i = \langle B_i, P_i \rangle$ be the blockchain view at node p_i in system S. For a transaction*

[4]In theory, there cannot be consensus on a block at a partiular index [16], hence preventing persistence, however, applications have successfully used Ethereum to transfer digital assets based on parameter $m_{ethereum} = 11$ [35].

tx to be locally committed *at p_i, the conjunction of the following properties must hold in p_i's view ℓ_i:*

1) *Transaction tx has to be in a block $b_0 \in B_i$ of the main branch of system S. Formally, $tx \in b_0 \wedge b_0 \in B'_i : \langle B'_i, P'_i \rangle = \text{get-main-branch}()_i$.*
2) *There should be a subsequence of m blocks $b_1, ..., b_m$ appended after block b. Formally, $\exists b_1, ..., b_m \in B_i : \langle b_1, b_0 \rangle, \langle b_2, b_1 \rangle, ..., \langle b_m, b_{m-1} \rangle \in P_i$. (In short, we say that b_0 is* decided*.)*

A transaction tx is committed *if there exists a node p_i such that tx is* locally committed*.*

Property (1) is needed because nodes eventually agree on the main branch that defines the current state of accounts in the system—blocks that are not part of the main branch are ignored. Property (2) is necessary to guarantee that the blocks and transactions currently in the main branch will persist and remain in the main branch. Before these additional blocks are created, nodes may not have reached consensus regarding the unique blocks b at index j in the chain. This is illustrated by the fork of Figure 1 where nodes consider, respectively, the pointer $\langle b_1, g \rangle$ and the pointer $\langle b_2, g \rangle$ in their local blockchain view. By waiting for m blocks were m is given by the blockchain system, the system guarantees with a reasonably high probability that nodes will agree on the same block b. Note that the property is not prefix-closed in the sense that get-main-branch may return a chain that is not necessarily a prefix of another branch returned later.

Algorithm 4 Checking transaction commit at node p_i

```
19: is-committed(tx)_i:                          ▷ check whether transaction is committed
20:     ⟨B'_i, P'_i⟩ ← get-main-branch()         ▷ pick main branch with Alg. 2 or 3
21:     if ∃b_0 ∈ B'_i : tx ∈ b_0 ∧ ∃b_1, ..., b_m ∈ B_i :    ▷ tx in main branch
22:           ⟨b_1, b_0⟩, ⟨b_2, b_1⟩..., ⟨b_m, b_{m-1}⟩ ∈ P_i then    ▷ enough blocks
23:         return true
24:     else return false
```

For example, consider a fictive blockchain system with $m_{fictive} = 2$ that selects the heaviest branch (Alg. 3, lines 8–15) as its main branch. If the blockchain state was the one depicted in Figure 2, then blocks b_2 and b_5 would be decided and all their transactions would be committed. This is because they are both part of the main branch and they are followed by at least 2 blocks, b_8 and b_{13}. (Note that we omit the genesis block as it is always considered decided but does not include any transaction.)

III. The Balance Attack

In this section, we present the Balance attack, a novel form of attacks that affect proof-of-work blockchains, especially Ethereum. Its novelty lies in identifying subgroups of miners of equivalent mining power and delaying messages between them rather than entering a race to mine blocks faster than others.

The Balance attack demonstrates a fundamental limitation of main proof-of-work systems in that they are not immutable.

Definition 2 (Corruptibility)**.** *A blockchain system is* corruptible *if an adversary can:*

1) *make the recipient of a transaction tx observe that tx is committed and*
2) *later remove the transaction tx from the main branch, with probability $1 - \varepsilon$.*

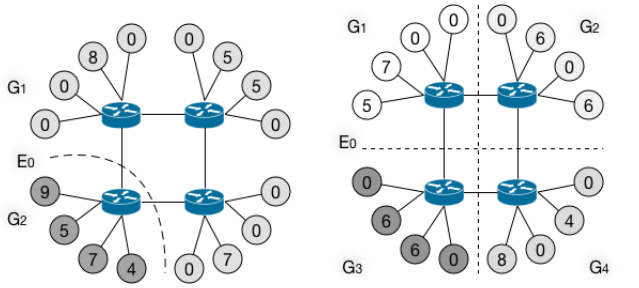
The Balance attack is simple: while the attacker disrupts communications between correct subgroups of equivalent mining power, it simply issues transactions in one subgroup. The attacker then mines sufficiently many blocks in another subgroup to ensure with high probability that the subtree of another subgroup outweighs the transaction subgroup's. Even though the transactions are committed, the attacker can rewrite with high probability the blocks that contain these transactions by outweighing the subtree containing this transaction.

Note that one could benefit from delaying messages only between the merchant and the rest of the network by applying the eclipse attack [22] to Ethereum. Eclipsing one node of Bitcoin appeared, however, sufficiently difficult: it requires to restart the node's protocol in order to control all the logical neighbors the node will eventually try to connect to. While a Bitcoin node typically connects to 8 logical neighbors, an Ethereum node typically connects to 25 nodes, making the problem even harder. Another option would be to isolate a subgroup of smaller mining power than another subgroup, however, it would make the attack only possible if the recipients of the transactions are located in the subgroup of smaller mining power. Although possible this would limit the generality of the attack, because the attacker would be constrained on the transactions it can override.

Note that the Balance attack inherently violates the persistence of the main branch prefix and is enough for the attacker to double spend. The attacker has simply to identify the subgroup that contains merchants and create transactions to buy goods from these merchants. After that, it can issue the transactions to this subgroup while propagating its mined blocks to at least one of the other subgroups. Once the merchant shipped goods, the attacker stops delaying messages. Based on the high probability that the tree seen by the merchant is outweighed by another subtree, the attacker could reissue another transaction transferring the exact same coin again.

A. Executing a Balance Attack

For the sake of simplicity, let us fix $k = 2$ and postpone the general analysis for any $k \geq 2$ to the companion technical report [34]. We consider subgraphs $G_1 = \langle V_1, E_1 \rangle$ and $G_2 = \langle V_2, E_2 \rangle$ of the communication graph $G = \langle V, E \rangle$ so that each subgraph has half of the mining power of



(a) Example of the selection of edges E_0 delayed between $k = 2$ subgraphs of 25 units of mining power each

(b) Example of the selection of edges E_0 delayed between $k = 4$ subgraphs of 12 units of mining power each

Figure 3: Two decompositions of communication graphs into subgraphs by an attacker where E_0 represents the cut of communication edges linking the subgraphs

Algorithm 5 The Balance attack initiated by attacker p_i

1: **State:**
2: $G = \langle V, E \rangle$, the communication graph
3: pow, a mapping from of a node in V to its mining power in $\mathbb{R}$
4: $\ell_i = \langle B_i, P_i \rangle$, the local blockchain at node p_i is a directed acyclic
5: graph of blocks B_i and pointers P_i
6: $\rho \in (0; 1)$, the portion of the mining power of the system owned
7: by the attacker p_i, with $0 < \rho < 0.5$
8: d, the difficulty of the crypto-puzzle currently used by the system

9: balance-attack$(\langle V, E \rangle)_i$: ▷ *starts the attack*
10: Select $k \geq 2$ subgraphs $G_1 = \langle V_1, E_1 \rangle, ..., G_k = \langle V_k, E_k \rangle$:
11: $\sum_{\forall v \in V_1} \mathsf{pow}(v) \approx ... \approx \sum_{\forall v' \in V_k} \mathsf{pow}(v')$
12: Let $E_0 = E \setminus \cup_{\forall 0 < i \leq k} E_i$ ▷ *attack communication channels*
13: Stop communications on E_0 during $\tau \geq \frac{(1-\rho)6d\log(\frac{4}{\varepsilon})}{\rho^2 t}$ seconds
14: Issue tx crediting a merchant in graph G_i with coins C
15: Let b_2 be a block appearing in G_j but not in G_i
16: Start mining on ℓ_i immediately after b_2 ▷ *contributed to correct chain*
17: Send blockchain view ℓ_i to some subgraph G_j where $j \neq i$
18: When τ seconds have elapsed, stop delaying messages on E_0
19: Issue transaction tx' that double spends coins C

the system as depicted in Figure 3(a) whereas Figure 3(b) illustrates the variant when $k = 4$. Let $E_0 = E \setminus (E_1 \cup E_2)$ be the set of edges that connects nodes of V_1 to nodes of V_2 in the original graph G. Let τ be the communication delay introduced by the attacker on the edges of E_0.

As indicated in Algorithm 5, the attacker can introduce a sufficiently long delay τ during which the miners of G_1 mine in isolation of the miners of G_2 (line 13). As a consequence, different transactions get committed in different series of blocks on the two blockchains locally viewed by the subgraphs G_1 and G_2. Let b_2 be a block present only in the blockchain viewed by G_2 but absent from the blockchain viewed by G_1. In the meantime, the attacker issues transactions spending coins C in G_1 (line 14) and mines a blockchain starting from the block b_2 (line 16). Before the delay expires the attacker sends his blockchain to G_2. After the delay expires, the two local views of the blockchain are exchanged. Once the heaviest branch that the attacker contributed to is adopted, the attacker can simply reuse the coins C in new transactions (line 19).

B. Exploiting the knowledge about the network

As indicated by the state of Algorithm 5, an attacker has to be knowledgeable about the current settings of the blockchain system to execute a Balance attack. In fact, the attacker must have information regarding the logical or physical communication graph, the mining power of the miners or pools of miners and the current difficulty of the crypto-puzzle.

The dynamic information regarding the mining power and the difficulty of nodes is generally public information and can often be retrieved online. In particular, the propagation delays, the list of connected nodes and their individual mining power, the version of their Ethereum client as well as the current difficulty of the crypto-puzzle can be found at https://ethstats.net/.

It is more difficult, however, to gather information regarding the communication network. Note that some tools exist to retrieve information regarding the communication topology of blockchain systems. The interesting aspects of the Balance attack is that it can apply to the logical overlay used by the peer-to-peer network of the blockchain system or to the physical overlay. While there exist tools to retrieve the logical overlay topology, like AddressProbe [28] to find some information regarding the peer-to-peer overlay of Bitcoin, it can be easier for an attacker of Ethereum to run a man-in-the-middle attack rather than a BGP hijacking [25], [1].

C. Delaying networking communications

While disrupting the communication between subgroups of a blockchain system may look difficult, there have been multiple attacks successfully delaying blockchains messages in the past.

In 2014, a BGP hijacker exploited access to an ISP to steal $83000 worth of bitcoins by positioning itself between Bitcoin pools and their miners [25]. Some known attacks against Bitcoin involved partitioning the communication graph at the network level [1] and at the application level [22]. At the network level, a study indicated the simplicity for autonomous systems to intercept a large amount of bitcoins and evaluated the impact of these network attacks on the Bitcoin protocol [1]. At the application level, some work showed that an attacker controlling 32 IP addresses can "eclipse" a Bitcoin node with 85% probability [22].

Starting in September 2016, Ethereum experienced a denial-of-service attacks that forced miners to spend a long time accessing memory while executing smart contracts. While it did not lead to double-spending as far as we know it

slowed down the entire network.[5] More generally, there are varieties of network attacks, known as man-in-the-middle attacks and spoofing attacks, that can be exploited to lead to similar results by relaying the traffic between two nodes through the attacker.

IV. Vulnerability of the Ghost Protocol

In this Section, we show that the Balance attack makes a blockchain system based on GHOST (depicted in Alg. 3) corruptible. A malicious user can issue a Balance attack with less than half of the mining power by delaying the network. Let us first summarize previous and new notations in Table I.

t	total mining power of the system (in MH/s)
d	difficulty of the crypto-puzzle (in MH)
ρ	fraction of the mining power owned by the adversary (in %)
k	the number of communication subgraphs
τ	disruption time of communication between subgraphs (seconds)
μ_c	mean of the number of blocks mined by a subgraph during τ
μ_m	mean of the number of blocks mined by the attacker during τ
Δ	the maximum difference of mined blocks for the two subgraphs

Table I: Notations of the analysis

For the sake of simplicity, we assume that $k = 2$ and $\sum_{\forall v \in V_1} \mathsf{pow}(v) = \sum_{\forall v' \in V_2} \mathsf{pow}(v')$ so that the communication is delayed between only two communication subgraphs of equal mining power. We defer the proofs to the companion technical report [34].

As there is no better strategy for solving the crypto-puzzles than random trials, we consider that subgraphs G_1 and G_2 mine blocks during delay τ. During that time, each of G_1 and G_2 performs a series of $n = \frac{1-\rho}{k} t\tau$ independent and identically distributed Bernoulli trials, each returning one in case of success with probability $p = \frac{1}{d}$ and 0 otherwise. Let the sum of these outcomes for subgraphs G_1 and G_2 be the random variables X_1 and X_2, respectively, each with a binomial distribution and mean:

$$\mu_c = np = \frac{(1-\rho)t\tau}{2d}. \quad (1)$$

Similarly, the mean of the number of blocks mined by the malicious miner during time τ is

$$\mu_m = \frac{\rho t \tau}{d}.$$

The Balance attack relies on the adversary outnumbering the difference in blocks mined by each subgraph G_1 and G_2. Thus, we first lower-bound the probability $\Pr[\mu_m > \Delta]$ that the expected number of blocks μ_m mined by the attacker is greater than the difference $\Delta = |X_1 - X_2|$ in blocks mined by the two subgraphs G_1 and G_2.

Theorem 3. *At time τ, the probability $\Pr[\mu_m > \Delta]$ that the expected number of blocks μ_m mined by the attacker is greater than the difference $\Delta = |X_1 - X_2|$ in blocks mined by the two subgraphs G_1 and G_2 is $\Pr[\mu_m > \Delta] > 1 - 4e^{-\frac{\rho^2}{3(1-\rho)^2}\mu_c}$.*

Corollary 4. *A blockchain system that selects a main branch based on the* GHOST *protocol (Alg. 3) is corruptible.*

Proof. By lines 17–18 of Alg. 3, we know that GHOST counts the mined blocks to compute the weight of a subtree, and to select one blockchain view and discard the other.

Since by Theorem 3, the expected number of blocks mined by the attacker at time τ is greater than the difference Δ with probability larger than $1 - 4e^{-\frac{\rho^2}{3(1-\rho)^2}\mu_c} \geq 1 - \varepsilon$, we know that when the timer expires at line 18 of Alg. 5, the attacker can make the system discard the blockchain view of either G_1 or G_2 with probability larger than $1-\varepsilon$ by sending its blockchain view to the other subgraph, hence making the blockchain system corruptible. ■

V. Analysis of the R3 Network

The statistics of the R3 network were gathered through the `eth-netstat` applications at the end of June 2016. R3 is a consortium of more than 50 banks that has tested blockchain systems and in particular Ethereum in a consortium private chain context over 2016.[6] The network consisted at that time of $|V| = 50$ nodes among which only 15 were mining. The mining power of the system was about 20 MH/s, the most powerful miner mined at 2.4 MH/s or 12% of the total mining power while the difficulty of the crypto-puzzle was observed close to 30 MH.

First, let assume that the attacker is the `r3` node with $\rho = 12\%$ of the mining power as depicted in Figure 4 and that it delays communication between subgraphs G_1 and G_2, each with mining power $\frac{1-\rho}{2} t = 8.8$ MH/s. The probability p of solving the crypto-puzzle per hash tested is $\frac{1}{30\times10^6}$ so that the mean is $\mu_c = \frac{(1-\rho)t\tau}{2d} = \frac{8.8\times10^6\times1180}{2\times30\times10^6} = 346.13$ if we wait for 19 minutes and 40 seconds, i.e., 1180 seconds. The attacker creates, in expectation, a block every $\frac{30}{2.4} = 12.5$ seconds or $\lfloor \frac{1180}{12.5} \rfloor = 94$ blocks during the 19 minutes and 40 seconds. Hence let $\delta = \rho/(1-\rho) = 0.136$. The probability that the attack is a success is 53%.

Second, let assume that the adversary controls $\rho = \frac{1}{3}$ of the total mining power, which represents $\rho t = 6.7$ MH/s. In this case the adversary delay communications between two subgraphs G_1 and G_2 with mining power of $\frac{1-\rho}{2} t = 6.7$ MH/s each. If we wait for 4 minutes, i.e., 240 seconds, then each isolated graph and adversary would mine $6.67 * 10^6 * 240/(30 * 10^6) = 53.4$ blocks. The probability that the attack succeeds would become $1 - e^{-\rho^2/3(1-\rho)^2 * 53.4}$, which is around 94%.

[5] https://blog.ethereum.org/2016/09/22/ethereum-network-currently-undergoing-dos-attack/.

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

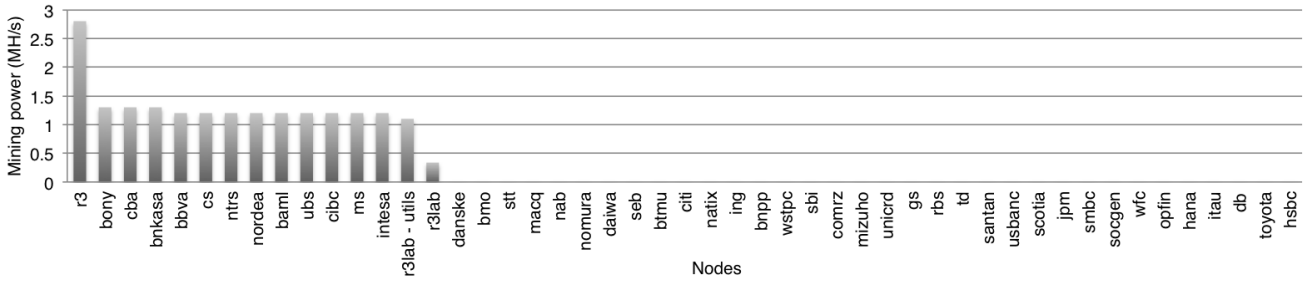


Figure 4: The mining power of the R3 Ethereum network as reported by `eth-netstats` as of June 2016

VI. Experimenting the Balance Attack on a Private Ethereum Blockchain

In this section, we experimentally produce the attack on an Ethereum private chain involving up to 18 physical distributed machines. To this end, we configure a realistic network with 15 machines dedicated to mining as in the R3 Ethereum testnet and 3 dedicated network switches.

All experiments were run on 18 physical machines of the Emulab environment where a network topology was configured using ns/2. The topology consists of three local area networks configured through a ns/2 configuration file with 20 ms latency and 100 Mbps bandwidth. All miners run the `geth` Ethereum client v.1.3.6 and the initial difficulty of the crypto-puzzle is set to 40 KH. The communication graph comprises the subgraph G_1 of 8 miners that includes the attacker and 7 correct miners and a subgraph G_2 of 7 correct miners.

A. Favoring one blockchain view over another

We run our first experiment during 2 minutes. We delayed the link E_0 during 60 seconds so that both subgraphs mine in isolation from each other during that time and end up with distinct blockchain views. After the delay we take a snapshot, at time t_1, of the blocks mined by each subgraphs and the two subgraphs start exchanging information normally leading to a consensus regarding the current state of the blockchain. At the end of the experiment, after 2 minutes, we take another snapshot t_2 of the blocks mined by each subgraph.

	# blocks at t_1	# blocks discarded at t_2	# blocks kept at t_2	retention
G_1	52	39	13	25%
G_2	58	58	0	0%

Table II: Number of blocks in the main branch (excluding uncles) mined by the subgraphs G_1 and G_2; the adversary influences the selection of branches and keeps blocks from G_1 but discards blocks from G_2

Table II lists the number of blocks (excluding uncles) of the blockchain views of G_1 and G_2 at times t_1, while the two subgraphs did not exchange their view, and at time t_2, after the subgraphs exchanged their blocks. Note that we did not represent the uncle blocks to focus on the main branches. We observe that the blockchain view of the subgraph G_1 was adopted as the valid chain while the other blockchain view of the subgraph G_2 was not. In particular, we retrieved 13 blocks of the main branch of G_1 at time t_1 in the main branch selected at t_2. As expected, all the blocks of G_2 at time t_1 were discarded from the main branch by time t_2.

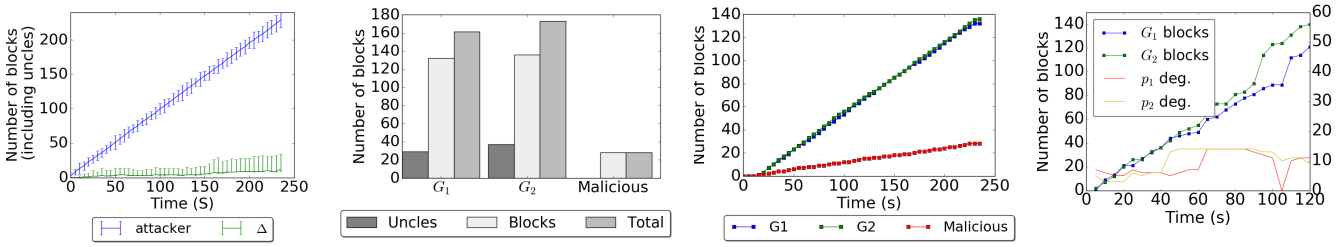
B. Blocks mined by an attacker and two subgraphs

We now report the total number of blocks mined, especially focusing on the creation of uncle blocks. More precisely, we compare the number of blocks mined by the attacker against the difference of the number of blocks Δ mined by each subgraph. We know from the analysis that it is sufficient for the attacker to mine at least $\Delta + 1$ blocks in order to be able to discard one of the k blockchain views, allowing for double spending. The experiment is similar to the previous experiment in that we also used Emulab with the same ns/2 topology, however, we did not introduce delays and averaged results over 10 runs of 4 minutes each.

Figure 5(a) depicts the minimum, maximum and average blocks obtained over the 10 runs. The vertical bars indicate minimum and maximum. First, we can observe that the average difference Δ is usually close to its minimum value observed during the 10 runs. This is due to having a similar total number of blocks mined by each subgraph in most cases with few rare cases where the difference is larger. As we can see, the total number of blocks (including uncles) mined during the experiment by the attacker is way larger than the difference in blocks Δ mined by the two subgraphs. This explains the success of the Balance attack as was observed in Section VI-A.

C. The role of uncle blocks in Ethereum

In the previous experiment, we focused on the total number of blocks without differentiating the blocks that are adopted in the main branch and the uncle blocks that are only part of the local blockchain views. The GHOST protocol accounts for these uncle blocks to decide the current state of the blockchain as we explained previously in Section II.



(a) Number of blocks mined by the attacker and the difference Δ in the number of blocks mined by the subgraphs

(b) Number of blocks mined by the attacker and the two subgraphs G_1 and G_2

(c) The depth of the blockchains mined by the attacker and two subgraphs G_1 and G_2

(d) Evolution of blocks and degrees where the attacker in G_1 delays the communication between G_1 and G_2 for one minute

Figure 5: Distributed experiments performed on a blockchain with 15 physical machines connected by a 100 Mbps network.

Figure 5(b) indicates the number of uncle blocks in comparison to the blocks accepted on the current state of the blockchain for subgraphs G_1 and G_2, and the attacker (referred to as 'Malicious'). As expected, we can observe that the malicious node does not produce any uncle block because he mines the block in isolation of the rest of the network, successfully appending each mined block consecutively to the latest block of its current blockchain view. We note several uncle blocks in the subgraphs, as correct miners may mine blocks concurrently at the same indices.

Figure 5(c) depicts the creation of the number of mined blocks (excluding uncle blocks) over time for subgraphs G_1 and G_2, and the attacker (referred to as 'Malicious'). As we can see the difference between the number of blocks mined on the subgraphs is significantly smaller than the number of blocks mined by the attacker. This explains why the Balance attack was observed in this setting.

D. Relating connectivity to known blocks

Figure 5(d) illustrates the execution of two subgraphs resolving connectivity issues and adopting a chain. This experiment outlines one of the fundamental aspects of the balance attack, in which the chosen subgraph resolves the network delay and attempts reconnection with another subgraph. At this point, the subgraphs will initiate the consensus protocol and select the branch to adopt as the main branch. The experiment was set up with two subgraphs G_1 and G_2 where $|V_1| = |V_2| = 7$. The attacker selects a subgraph and delays messages between this subgraph and another, enforcing an isolated mining environment. Once the delay is set, the attacker joins one of the subgraphs and begins to mine onto the current chain. The attacker then delays the messages until there is a sufficient amount of blocks mined onto the isolated blockchain for it to be adopted as the correct chain by the other subgraph. In this experiment, at $t = 60\,s$, the delay between subgraphs is resolved, and the subgraphs maintain a connection. Upon reconnection, the subgraphs invoke the consensus protocol to select and adopt the correct chain. In this case, using the GHOST protocol, the heaviest chain is selected for both subgraphs, meaning the chain mined by G_1 is chosen, to which the attacker contributed.

This result reveals that the adoption of a chosen blockchain is plausible, given that the attacker is able to sufficiently delay messages between subgraphs.

VII. Solution with Non-Forkable Blockchains

Corruptibility is a general problem that may affect other forkable blockchains, especially if they require messages to be propagated within a time bound, or may experience disagreement with a non-null probability [27], [13].

A. Other forkable blockchains

First, it is important to note that the corruptibility problem is not restricted to the GHOST algorithm or to the proof-of-work mechanism underlying Ethereum but could also apply to other forkable algorithms as well.

For example, we can slightly change the Balance attack to work in Bitcoin as well. While in Ethereum it was sufficient for the attacker to mine on any branch of the blockchain view of G_j after the block b_2 (Alg. 5, line 15), in Bitcoin the attacker has to mine at the top of the blockchain view of G_j. By doing so, the attacker increases the length of the Nakamoto's main branch in graph G_j. Considering that each correct miner mines at the top of the longest branch of their subgroup with the same probability q, the mean of the number of blocks added to the main branch will become $\mu_c^{bitcoin} = \frac{(1-\rho)t\tau}{2dq}$. We can then define two binomial random variables X_i' and X_j' for the expected number of blocks in the main branch of G_i and G_j, respectively, and apply a similar argument as in Section IV.

The absence of proof-of-work does not imply immunity against the Balance attack. Casper [6] has been proposed as a proof-of-stake alternative to GHOST for the Ethereum

blockchain. Even with Casper, the Ethereum blockchain would remain forkable. Validators are incentivized to bet some coin amount, called *value-at-loss*, on the block that they believe will be decided first among multiple ones. Once the aggregated value-at-loss bet by all validiators for a particular block reaches a threshold, then the block gets decided and the winning gambler gets rewarded. For the sake of availability, this threshold has to be adjusted depending on the responding participants. Like in the Balance attack, an adversary could delay communication between groups of nodes to alter the responsiveness of the participants and the threshold definition. By contrast with proof-of-work, the adversary would have to exploit some value-at-loss rather than its computational power to outweigh the bets and influence the selection of the main branch.

B. Non-forkable Blockchains

The crux of the problem of corruptible blockchains is that they favor availability over consistency. The well-known CAP theorem, initially mentioned by Brewer at PODC [4] and proved by Gilbert and Lynch [19], stating the impossibility for a distributed service to provide *(i)* consistency: returning the right response to a request; *(ii)* availability: returning a response to each request; and *(iii)* partition-tolerance: supporting message delays and losses. Intuitively, it appears natural for a distributed system to stop working as soon as communication is no longer ensured between the distributed participants. What is less clear is the consequence of communication being delayed. As we presented here, the consequence for forkable blockchains can be asset losses.

There exist a large body of solutions in the distributed computing literature favoring consistency over availability. As consortium and private blockchains involve a typically known number n of participants, Byzantine consensus algorithms could be used to cope with the Balance attack provided that strictly less than $\frac{n}{3}$ processes are under the control of the adversary. Several blockchains tried to build upon the classic Byzantine consensus implementation PBFT [8]. This is the case of Hyperledger[7], Tendermint [23] and Corda [5]. Unfortunately, PBFT does not perform well with a growing set of participants [10]. Hierarchical approaches were proposed to improve the performance of PBFT [26], but assume synchrony. Another tentative [29] improved performance by building upon a more recent Byzantine consensus algorithm [30], however, this algorithm is probabilistic [31]. A new deterministic Byzantine algorithm designed especially for blockchains has just been proposed [9]. Although it appears efficient and is guaranteed to terminate, to our knowledge it has not been evaluated empirically.

VIII. Related work

Traditional attacks against Bitcoin consist of waiting for some external action, like shipping goods, in response to a transaction before discarding the transaction from the main branch. As the transaction is revoked, the issuer of the transaction can reuse the coins of the transaction in another transaction. As the side effects of the external action cannot be revoked, the second transaction appears as a "double spending".

Perhaps the most basic form of such an attack assumes that an application takes an external action as soon as a transaction is included in a block [15], [24], [2]. The first attack of this kind is called Finney's attack and consists of solo-mining a block with a transaction that sends coins to itself without broadcasting it before issuing a transaction that double-spends the same coin to a merchant. When the goods are delivered in exchange of the coins, the attacker broadcasts its block to override the payment of the merchant. The vector76 attack [40] consists of an attacker solo-mining after block b_0 a new block b_1 containing a transaction to a merchant to purchase goods. Once another block b'_1 is mined after b_0, the attacker quickly sends b_1 to the merchant for an external action to be taken. If b'_1 is accepted by the system, the attacker can issue another transaction with the coins spent in the discarded block b_1.

The attacks become harder if the external action is taken after the transaction is committed by the blockchain. Rosenfeld's attack [38] consists of issuing a transaction to a merchant. The attacker then starts solo-mining a longer branch while waiting for m blocks to be appended so that the merchant takes an external action in response to the commit. The attack success probability depends on the number m of blocks the merchant waits before taking an external action and the attacker mining power [18]. However, when the attacker has more mining power than the rest of the system, the attack, also called *majority hashrate attack* or *51-percent attack*, is guaranteed successful, regardless of the value m. To make the attack successful when the attacker owns only a quarter of the mining power, the attacker can incentivize other miners to form a coalition [14] until the coalition owns more than half of the total mining power.

Without a quarter of the mining power, discarding a committed transaction in Bitcoin requires additional power, like the control over the network. It is well known that delaying network messages can impact Bitcoin [11], [37], [39], [20], [36]. Decker and Wattenhoffer already observed that Bitcoin suffered from block propagation delays [11]. Godel et al. [20] analyzed the effect of propagation delays on Bitcoin using a Markov process. Garay et al. [17] investigated Bitcoin in the synchronous communication setting, however, this setting is often considered too restrictive [7]. Pass et al. extended the analysis for when the bound on message delivery is unknown and showed in their model that the difficulty of Bitcoin's crypto-difficulty has to be adapted depending on the bound on the communication delays [37]. This series of work reveal an important limitation of Bitcoin: delaying propagation of blocks can waste the computational

[7] http://www.hyperledger.org/.

effort of correct nodes by letting them mine blocks unnecessarily at the same index of the chain. In this case, the attacker does not need more mining power than the correct miners, but simply needs to expand its local blockchain faster than the growth of the longest branch of the correct blockchain.

Ethereum proposed the GHOST protocol to cope with this issue [39]. The idea is simply to account for the blocks proposed by correct miners in the multiple branches of the correct blockchain to select the main branch. As a result, Growing a branch the fastest is not sufficient for an attacker of Ethereum to be able to double spend. Even though the propagation strategy of Ethereum differs from the pull strategy of Bitcoin, some network attacks against Bitcoin could affect Ethereum. In the Eclipse attack [22] against Bitcoin, the attacker forces the victim to connect to 8 of its malicious identities. The Ethereum adaptation would require to forge $3\times$ more identities and force as many connections as the default number of clients is 25. Apostolaki et al. [1] proposed a BGP hijacking attack and showed that the number of Internet prefixes that need to be hijacked for the attack to succeed depends on the distribution of the mining power. BGP-hijacking typically requires the control of network operators but is independent from Bitcoin and could potentially be exploited to delay network messages and execute a Balance attack in Ethereum.

Some work already evoked the danger of using proof-of-work techniques in a consortium context [21]. In particular, experiments demonstrated the impossibility of ordering even committed transactions in an Ethereum private chain without exploring the impact of the network delay [35]. We go further by showing that forkable blockchains may suffer from corruptibility.

IX. Conclusion

In this paper, we propose the Balance attack a new attack that combines mining power with communication delay to affect prominent forkable blockchain protocols like Ethereum and Bitcoin. This attack simply consists of convincing correct nodes to disregard particular proposed series of blocks to lead to a double spending. We analyzed the tradeoff inherent to Ethereum between communication delay and mining power, hence confirming that forkable designs are ill-suited for consortium and private blockchains.

There are several ways to extend this work. First, the context is highly dependent on medium-scale settings where statistics about all participants can be easily collected. It would be interesting to extend these results when the mining power of participants is unknown. Second, the success of the Balance attack despite a low mining power requires communication delay between communication subgraphs. The next step is to compare denial-of-service and man-in-the-middle attacks and evaluate their effectiveness in introducing this delay.

Acknowledgements

We are grateful to the R3 consortium for sharing information regarding their Ethereum testnet and to Seth Gilbert and Tim Swanson for comments on an earlier version of this paper.

References

[1] M. Apostolaki, A. Zohar, and L. Vanbever, "Hijacking bitcoin: Large-scale network attacks on cryptocurrencies," arXiv, Tech. Rep. 1605.07524, 2016.

[2] T. Bamert, C. Decker, L. Elsen, R. Wattenhofer, and S. Welten, "Have a snack, pay with bitcoins," in *13th IEEE International Conference on Peer-to-Peer Computing, IEEE P2P 2013, Trento, Italy, September 9-11, 2013, Proceedings*, 2013, pp. 1–5.

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

[4] E. Brewer, "Towards robust distributed systems," in *Proc. 19th ACM Symposium on Principles of Distributed Computing (PODC)*, July 2000.

[5] R. G. Brown, J. Carlyle, I. Grigg, and M. Hearn, "Corda: An introduction," 2016.

[6] V. Buterin, "Ethereum 2.0 mauve paper," in *Ethereum Developer Conference 2*, September 2016.

[7] C. Cachin, "Distributing trust on the internet," in *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, 2001, pp. 183–192.

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

[9] T. Crain, V. Gramoli, M. Larrea, and M. Raynal, "(leader/randomization/signature)-free byzantine consensus for consortium blockchains," arXiv, Tech. Rep. 1702.03068, 2017.

[10] 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.

[11] C. Decker and R. Wattenhofer, "Information propagation in the bitcoin network," in *Proc. of the IEEE International Conference on Peer-to-Peer Computing*, 2013.

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

[13] 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.

[14] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," in *Proceedings of the 18th Int'l Conference Financial Cryptography and Data Security (FC)*, 2014, pp. 436–454.

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

[16] 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.

[17] J. A. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *34th Annual Int'l Conf. on the Theory and Applications of Crypto. Techniques*, 2015, pp. 281–310.

[18] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2016, pp. 3–16.

[19] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *ACM SIGACT News*, vol. 33, pp. 51–59, 2002.

[20] J. Göbel, H. Keeler, A. Krzesinski, and P. Taylor, "Bitcoin blockchain dynamics: The selfish-mine strategy in the presence of propagation delay," *Performance Evaluation*, Juy 2016.

[21] V. Gramoli, "On the danger of private blockchains," in *Workshop on Distributed Cryptocurrencies and Consensus Ledgers (DCCL'16)*, 2016.

[22] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on bitcoin's peer-to-peer network," in *24th USENIX Security Symposium*, 2015, pp. 129–144.

[23] K. J., "Tendermint: Consensus without mining v.0.7," 2016.

[24] 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.

[25] P. Litke and J. Stewart, "BGP hijacking for cryptocurrency profit," August 2014. [Online]. Available: https://www.secureworks.com/research/bgp-hijacking-for-cryptocurrency-profit

[26] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 17–30.

[27] S. Micali, "Algorand: The efficient and democratic ledger. corr abs/1607.01341," arXiv, Tech. Rep. 1607.01341, 2016.

[28] A. Miller, J. Litton, A. Pachulski, N. Gupta, D. Levin, N. Spring, and B. Bhattacharjee, "Discovering Bitcoin's network topology and influential nodes," University of Maryland, Tech. Rep., 2015.

[29] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of bft protocols," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2016, pp. 31–42.

[30] A. Mostéfaoui, H. Moumen, and M. Raynal, "Signature-free asynchronous byzantine consensus with $T < N/3$ and $O(N^2)$ messages," in *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing*, ser. PODC '14. New York, NY, USA: ACM, 2014, pp. 2–9. [Online]. Available: http://doi.acm.org/10.1145/2611462.2611468

[31] ——, "Signature-free asynchronous binary byzantine consensus with $T < N/3$, $O(N^2)$ messages, and $O(1)$ expected time," *J. ACM*, vol. 62, no. 4, pp. 31:1–31:21, Sep. 2015.

[32] R. Motwani and P. Raghavan, *Randomized Algorithms*. Cambridge University Press, 1995.

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

[34] C. Natoli and V. Gramoli, "The balance attack against proof-of-work blockchains: The r3 testbed as an example," arXiv, Tech. Rep. 1612.09426, 2016.

[35] ——, "The blockchain anomaly," in *Proceedings of the 15th IEEE International Symposium on Network Computing and Applications (NCA'16)*, Oct 2016.

[36] K. Nayak, S. Kumar, A. Miller, and E. Shi, "Stubborn mining: Generalizing selfish mining and combining with an eclipse attack," in *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*, 2016, pp. 305–320.

[37] R. Pass, L. Seeman, and A. Shelat, "Analysis of the blockchain protocol in asynchronous networks," Crytology ePrint Archive, Tech. Rep. 454, 2016.

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

[39] 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.

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

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