PHANTOM and GHOSTDAG

A Scalable Generalization of Nakamoto Consensus

Yonatan Sompolinsky
yoni_sompo@cs.huji.ac.il
School of Engineering and Computer Science, The Hebrew University of Jerusalem, Israel
DAGlabs

Shai Wyborski
shaide@cs.huji.ac.il
School of Engineering and Computer Science, The Hebrew University of Jerusalem, Israel
DAGlabs

Aviv Zohar
avivz@cs.huji.ac.il
School of Engineering and Computer Science, The Hebrew University of Jerusalem, Israel

ABSTRACT

In 2008 Satoshi Nakamoto invented the basis for blockchain-based distributed ledgers. The core concept of this system is an open and anonymous network of nodes, or *miners*, which together maintain a public ledger of transactions. The ledger takes the form of a chain of blocks, *the blockchain*, where each *block* is a batch of new transactions collected from users. One primary problem with Satoshi's blockchain is its highly limited scalability. The security of Satoshi's *longest chain rule*, more generally known as *the Bitcoin protocol*, requires that all honest nodes be aware of each other's blocks very soon after the block's creation. To this end, the throughput of the system is artificially suppressed so that each block fully propagates before the next one is created, and that very few "orphan blocks" that fork the chain be created spontaneously.

In this paper we present PHANTOM, a proof-of-work based protocol for a permissionless ledger that generalizes Nakamoto's blockchain to a direct acyclic graph of blocks (blockDAG). PHANTOM includes a parameter k that controls the level of tolerance of the protocol to blocks that were created concurrently, which can be set to accommodate higher throughput. It thus avoids the security-scalability tradeoff which Satoshi's protocol suffers from.

PHANTOM solves an optimization problem over the blockDAG to distinguish between blocks mined properly by honest nodes and those created by non-cooperating nodes who chose to deviate from the mining protocol. Using this distinction, PHANTOM provides a robust total order on the blockDAG in a way that is eventually agreed upon by all honest nodes. Implementing PHANTOM requires solving an NP-hard problem, and to avoid this prohibitive computation, we devised an efficient greedy algorithm GHOSTDAG that captures the essence of PHANTOM.

We provide a formal proof of the security of GHOSTDAG, namely, that its ordering of blocks is irreversible up to an exponentially negligible factor. We discuss the properties of GHOSTDAG and how it compares to other DAG based protocols.

1 INTRODUCTION

The security of the Bitcoin protocol relies on blocks propagating quickly to all miners in the network [5, 10, 15]. Block creation itself is slowed down via the requirement that each block contain a proof-of-work. For the Bitcoin protocol to be secure, block propagation must be faster than the typical time it takes the network together to create the next block. In order to guarantee this property, the creation of blocks in Bitcoin is regulated by the protocol to occur only once every 10 minutes, and the block size itself is limited to allow for fast transmission. As a result, Bitcoin suffers from a highly restrictive throughput on the order of 3-7 transactions per second (tps)[1].

The PHANTOM protocol. In this paper we present PHANTOM, a protocol that generalizes Nakamoto's longest chain protocol. While Bitcoin blocks each contain a hash of a single predecessor block in the chain they are extending which implies that blocks for a tree, PHANTOM structures blocks in a Directed Acyclic Graph, a *blockDAG*. Each block can thus include several hash references to predecessors. PHANTOM then provides a total ordering over all blocks and transactions, and outputs a consistent set of accepted transactions. Unlike the Bitcoin protocol, where blocks that are not on the main chain are discarded, PHANTOM incorporates all blocks in the blockDAG into the ledger, but places blocks that were created by attackers later in the order.

In rough terms, PHANTOM consists of a three-step procedure:

(1) Using the structure of the blockDAG, we recognize a set of well-connected blocks (we later refer to these as blue blocks); this procedure is used to exclude blocks created by misbehaving nodes and is the heart of the protocol: Blocks that either reference only old blocks from the DAG, or are withheld by their creator for some time, will be excluded from the set of blue blocks with high probability.
(2) We complete the DAG's naturally induced partial ordering to a full topological order in a way that favours blocks inside the selected cluster and penalizes those outside it.
(3) The order over blocks induces an order over transactions; transactions in the same block are ordered according to the order of their appearance in the block. We iterate over all transactions in this order, and accept each one that is consistent (according to the underlying consistency notion) with all transactions approved so far.

Propagation delay. The first step above is parameterized with k, which is a function of the assumed network delay diameter. In that, PHANTOM is similar to Nakamoto Consensus which assumes that an upper bound on the network's delay diameter. In fact, under low throughput we can set $k = 0$, in which case PHANTOM coincides with Nakamoto Consensus.

However, while Nakamoto Consensus suppresses the throughput and sets the block creation rate λ such that $D \cdot \lambda \ll 1$, PHANTOM does not impose an *a priori* constraint over λ. Instead, the throughput (in terms of λ and the block size) can be set to approach the network's capacity, and then k can be set after the fact to ensure

[1] The exact figure varies according to the size of a typical transaction, and changes if one includes protocol changes such as SegWit, Schnorr signatures, etc.

the safety of the protocol. This alleviates the security-scalability tradeoff that Nakamoto Consensus suffers. Still, increasing k does not come without cost, as we will discuss shortly.

GHOSTDAG. In its vanilla form, PHANTOM requires solving an NP-hard problem, and is therefore unsuitable for practical applications. Instead, we use the intuition behind PHANTOM to devise a greedy algorithm, GHOSTDAG, which can be implemented efficiently. We prove formally that GHOSTDAG is secure, in the sense that its ordering of blocks becomes exponentially difficult to reverse as time develops.

The main achievement of GHOSTDAG can be summarized as follows:

Theorem (Informal)**.** *Given two transactions tx_1, tx_2 that were published and embedded in the blockDAG at some point in time, the probability that GHOSTDAG's order between tx_1 and tx_2 changes over time decreases exponentially as time grows, even under a high block creation rate that is non-negligible relative to the network's propagation delay, assuming that a majority of the computational power is held by honest nodes.*

We will reformalize this theorem in Section 3, and provide a formal proof in Section A.

We now proceed to describe the PHANTOM and GHOSTDAG protocols more formally.

2 THE PHANTOM PROTOCOL

2.1 Preliminaries

The following terminology is used extensively throughout this paper. A DAG of blocks is denoted $G = (C, E)$, where C represents blocks and E represents the hash references to previous blocks. We frequently write $B \in G$ instead of $B \in C$. $past(B, G) \subset C$ denotes the subset of blocks reachable from B, and similarly $future(B, G) \subset C$ denotes the subset of blocks from which B is reachable; these are blocks that were provably created before and after B, correspondingly. *An edge in the DAG points back in time, from the new block to previously created blocks which it extends.* We denote by $anticone(B, G)$ the set of blocks outside $past(B, G)$ and $future(B, G)$ (excluding B itself); this is the set of blocks in the DAG which did not reference B (directly or indirectly via their predecessors) and were not referenced by B (directly or indirectly via B's predecessors).[2] Finally, $tips(G)$ is the set of blocks with in-degree 0 (usually, the most recent blocks). This terminology is demonstrated in Figure 1.

2.2 The DAG mining protocol

Rather than extending a single chain, a miner in PHANTOM references in its new block all blocks in $tips(G)$, where G is the DAG that the miner observes locally at the time when the new block is created. Additionally, the miner should broadcast its new block as fast as possible. These two rules together constitute the DAG mining protocol in PHANTOM.

[2]We will frequently abbreviate the notation and write, e.g., $anticone(B)$, instead of $anticone(B, G)$.

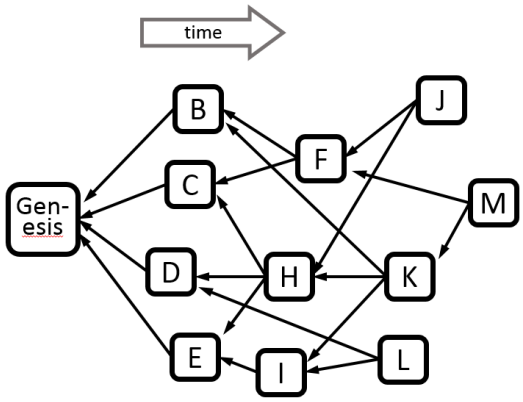

Figure 1: An example of a block DAG G. Each block references all blocks that were known to its miner at the time it was created. The DAG terminology applies to H as follows:
$past(H) = \{Genesis, C, D, E\}$ – blocks which H references directly or indirectly, and which were provably created before H;
$future(H) = \{J, K, M\}$ – blocks which reference H directly or indirectly, and which were provably created after H;
$anticone(H) = \{B, F, I, L\}$ – the order between these blocks and H is ambiguous. Reaching consensus on the order between blocks and other blocks in their anticone is the main challenge that we face.
$tips(G) = \{J, L, M\}$ – leaf-blocks, namely, blocks with in-degree 0; these will be referenced in the header of the next block.

2.3 The DAG ordering protocol

The aforementioned DAG mining protocol implies in particular that even when two blocks contain conflicting transactions, both blocks are incorporated into the blockDAG and referenced by all (honest) miners. The core challenge is then how to recover the consistency of the blockDAG. This is done in our framework by ordering all blocks – and by extension, all transactions – and accepting transactions one by one, eliminating individual transactions that are inconsistent with those approved before them. PHANTOM achieves consensus on the order of blocks, and this guarantees agreement on the set of accepted transactions as well.

Essentially, Bitcoin can be seen as an ordering protocol as well, according to which transactions embedded in the longest chain of blocks precede those off the longest chain. Unfortunately, Bitcoin's protocol is known to be secure only under slow block rates (see Section 4).

The ordering rule of PHANTOM has two stages: First, we divide the blocks to Blues and Reds; the Blue set represents blocks that appear to have been mined by cooperating nodes, whereas blocks in the Red set are outliers that were most likely mined by malicious or strategic nodes. Then, we order the DAG in a way that favours blue blocks and penalized red ones. The latter step is rather immediate,

and the novelty of PHANTOM lies mainly in the first colouring procedure.

2.3.1 The intuition behind PHANTOM. Just like Bitcoin, PHANTOM relies on the ability of honest nodes to communicate to their peers recent blocks in a timely manner, and on the assumption that honest nodes possess more than 50% of the hashrate. The block rate in Bitcoin is suppressed so as to ensure block creation is slower than the time it takes to communicate them. In PHANTOM, on the other hand, we notice that the set of honest blocks can be recognized even when the block rate is high and many forks appear spontaneously: Due to the communication and cooperation of honest mines, we should expect to see in the DAG a "well-connected" cluster of blocks.

Indeed, let D be an upper bound on the network's propagation delay. If block B was mined by an honest miner at time t, then any block published before time $t - D$ necessarily arrived at its miner before time t, and is therefore included in $past(B)$. Similarly, the honest miner will publish B immediately, and so B will be included in the past set of any block mined after time $t+D$. As a result, the set of honest blocks in B's anticone is typically small, and consists only of blocks created in the interval $[t - D, t + D]$. The proof-of-work mechanism guarantees that the number of blocks created in an interval of length $2 \cdot D$ is typically below some $k > 0$.

In short, the set of blocks created by honest nodes is well-connected. The following definition captures "well-connectedness":

Definition 1. *Given a DAG $G = (C, E)$, a subset $S \subseteq C$ is called a k-cluster, if $\forall B \in S : |anticone(B) \cap S| \leq k$.*

Attacker nodes may deviate arbitrarily from the mining rules, have large anticones, and even artificially increase the anticone of honest blocks. Nonetheless, since honest miners possess more proof-of-work power, it is usually impossible for malicious miners to create a well-connected set of blocks that is larger than that created by honest nodes. PHANTOM utilizes this fact and selects the largest well-connected set within the DAG, by solving the following optimization problem:

Maximum k-cluster SubDAG (MCS_k)
Input: DAG $G = (C, E)$
Output: A subset $S^* \subset C$ of maximum size, s.t. $|anticone(B) \cap S^*| \leq k$ for all $B \in S^*$.

In this formulation, the parameter k is predetermined; see Section 4 for more details. An example of a maximum k-cluster appears in Figure 2.

2.3.2 The PHANTOM protocol. Following the above intuition, the ordering protocol of PHANTOM comprises the following two steps:

(1) Given a block G, solve $MCS_k(G)$; let's refer to its output as the Blue set and to its complement set as the Red one.
(2) Determine the order between Blue blocks according to some topological sort. Then, for any Blue block B, add to the order just before B all of the Red blocks in $past(B)$ that weren't added to the order yet; these Red blocks too should be added in a topological manner.[3]

[3]The topological sorts in this second step are additionally required to be agnostic to future blocks: For any Blue block B, the order on blocks in $past(B)$ should remain the same if we remove from the DAG blocks in $future(B)$. This can be implemented, for instance, using a priority queue that pops out blocks according to the size of their past set.

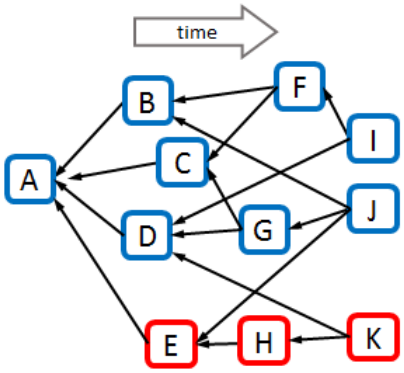

Figure 2: An example of the largest 3-cluster of blocks within a given DAG: A, B, C, D, F, G, I, J (coloured blue). It is easy to verify that each of these blue blocks has at most 3 blue blocks in its anticone, and (a bit less easy) that this is the largest set with this property. Setting PHANTOM's inter-connectivity parameter with $k = 3$ means that at most 4 blocks are assumed to be created within each unit of delay, so that typical anticone sizes should not exceed 3. Blocks outside the largest 3-cluster, E, H, K (coloured red), belong to the attacker (w.h.p.). For instance, block E has 6 blue blocks in its anticone (B, C, D, F, G, I); these blocks didn't reference E, presumably because E was withheld from their miners. Similarly, block K admits 6 blue blocks in its anticone (B, C, G, F, I, J); presumably, its malicious miner received already some blocks from (B, C, D, G), but violated the mining protocol by not referencing them.

An example of the output of the PHANTOM procedure on the small blockDAG from Figure 2 is: $(A, D, C, G, B, F, I, E, J, H, K)$. Unfortunately, the Maximum k-cluster SubDAG problem is NP hard (see problem [GT26] in [6]), and PHANTOM is therefore of less practical use for an ever-growing blockDAG. We thus introduce a greedy algorithm that is more suitable for implementation. We call this greedy variant GHOSTDAG.[4]

2.4 The GHOSTDAG protocol

Similar to PHANTOM, the GHOSTDAG protocol selects a k-cluster, which induces a colouring of the blocks as Blues (blocks in the selected cluster) and Reds (blocks outside the cluster). However, instead of searching for the largest k-cluster, GHOSTDAG finds a k-cluster using a greedy algorithm. The algorithm constructs the Blue set of the DAG by first inheriting the Blue set of the best tip $B_{\max}$, i.e., the tip with the largest Blue set in its past, and then adds

[4]In previous versions of this paper we referred to both versions as PHANTOM. Essentially, GHOSTDAG can be seen as a fix of the greedy algorithm introduced by the authors in [15], called the GHOST protocol. We thank *** *** for suggesting this name.

Yonatan Sompolinsky, Shai Wyborski, and Aviv Zohar

to the Blue set blocks outside $B_{\max}$'s past in a way that preserves the k-cluster property.

Observe that this greedy inheritance rule induces a chain: The last block of the chain is the selected tip of G, $B_{\max}$; the next block in the chain is the selected tip of the DAG $past\,(B_{\max})$; and so on down to the *genesis*. We denote this chain by $Chn(G) = (genesis = Chn_0(G), Chn_1(G), \ldots, Chn_h(G))$.

The final order over all blocks, in GHOSTDAG, follows a similar path as the colouring procedure: We order the blockDAG by first inheriting the order of B_{max} on blocks in $past\,(B_{max})$, then adding B_{max} itself to the order, and finally adding blocks outside $past\,(B_{max})$ according to some topological ordering. Thus, essentially, the order over blocks becomes robust as the colouring procedure.

2.4.1 Formal algorithm. The procedures described above are formalized in Algorithm 1 below. The algorithm begins with the base case where the DAG consists of the *genesis* block only (lines 2-3). Next, it performs a recursive call to compute the Blue sets and ordering of the past of each of the DAG's tips (lines 4-5), and inherits those of the best tip (lines 6-8). Then, the selected tip is added to the Blue set $BlueSet_G$ and to the last position in the current ordered list $OrderedList_G$ (lines 9-10). Then we iterate over $anticone\,(B_{\max}, G)$ in some topological way which guarantees that a block is visited only after its predecessors are (lines 11-14).[5] For every block we visit, we check if adding B to the Blue set will preserve the k-cluster property, and if this condition is satisfied, we add B to the Blue set (lines 9-13); either way, we add B to the current last position in the list (line 14). Finally, we return the Blue set and the ordered Note that the recursion in the algorithm (line 5) halts, because for any block $B \in G$: $|past\,(B)| < |G|$.

Algorithm 1 Ordering the DAG

Input: G – a block DAG, k – the propagation parameter
Output: $BLUE_k\,(G)$ – the Blue set of G; ord – an ordered list containing all blocks in G

1: **function** OrderDAG(G, k)
2: **if** $G == \{genesis\}$ **then**
3: **return** $[\{genesis\}, \{genesis\}]$
4: **for** $B \in tips(G)$ **do**
5: $[BlueSet_B, OrderedList_B] \leftarrow$OrderDAG($past\,(B), k$)
6: $B_{\max} \leftarrow \arg\max\{|BlueSet_B| : B \in tips(G)\}$ (break ties according to lowest hash)
7: $BlueSet_G \leftarrow BlueSet_{B_{\max}}$
8: $OrderedList_G \leftarrow OrderedList_{B_{\max}}$
9: add $B_{\max}$ to $BlueSet_G$
10: add $B_{\max}$ to the end of $OrderedList_G$
11: **for** $B \in anticone\,(B_{\max}, G)$ **do** in some topological ordering
12: **if** $BlueSet_G \cup \{B\}$ is a k-cluster **then**
13: add B to $BlueSet_G$
14: add B to the end of $OrderedList_G$
15: **return** $[BlueSet_G, OrderedList_G]$

[5]This can be implemented in several ways, e.g., by inserting all blocks in $anticone\,(B_{\max})$ into a deterministic priority queue which respects the topology. That is, the queue should pop out a block only after all of its parents have been popped out, and the order in which it pops blocks should be fully determined by the blockDAG.

We demonstrate the operation of this algorithm in Figure 3.We will introduce some variants of Algorithm 1 in Section 6. Finally, we have an efficient implementation of Algorithm 1, and we tested it under high block rates ($\lambda = 10$ and 25 blocks per second, for blocks of size 0.1-1 MB). We will make the implementation available in the full version of this paper.

2.5 Collapse to Bitcoin when $k = 0$

If the block creation rate is kept low, as in Bitcoin, the parameter k can be safely set to 0, as honest blocks are likely to create a chain (see Section 4). In such a setup, both PHANTOM and GHOSTDAG converge to Bitcoin's rule in the sense that, in case of a chain split, blocks on the longest chain precede those off the longest chain. These protocols would differ from Bitcoin's rule in that, as of according to the DAG mining protocol, blocks in the longest chain would eventually reference those off the longest chain, in PHANTOM or GHOSTDAG, and will thereby insert them in the order.

In fact, it is easy to see that the longest chain is, by definition, the largest 0-cluster of the DAG. Accordingly, our protocols can be seen as a generalization of Satoshi's longest-chain rule to a setup where block rate is high and propagation delays are not negligible.

In Appendix A we prove that GHOSTDAG remains secure under high block creation rates as well. We leave the formal analysis of PHANTOM for future work.

3 FORMAL MODEL AND STATEMENT

In this section we describe our formal framework. While we introduce new notation and terminology, the reader should keep in mind that *we stick to Nakamoto Consensus's model in almost every respect*—transactions, blocks, Proof-of-work, computationally bounded attacker, P2P propagation of blocks, probabilistic security guarantees, etc. The "only" difference is that a block references (possibly) several predecessors rather than a single one. While this has far reaching consequences on how the ledger is to be interpreted, on the mining side things remain largely the same.

3.1 Network

We follow the model specified in [14]. The network of nodes (or miners) is denoted $\mathcal{N}$, *honest* denotes the set of nodes that abide to the mining protocol (as defined below), and *malicious* denotes the rest of the nodes. Honest nodes form a connected component in $\mathcal{N}$'s topology, and the communication delay diameter of the honest subnetwork is D: if an honest node $v \in \mathcal{N}$ sends a message of size b MB at time t, it arrives at all honest nodes by time $t + D$ the latest. The attacker is assumed to suffer no delays on its outgoing or incoming links.

The real value of D is *a priori* unknown. The PHANTOM protocol assumes that D is always smaller than some constant D_{max} (both depend on the block size b). The parameter D_{max} is not hard-coded explicitly in the protocol, rather it influences another parameter, $k = k(D_{max})$, which is hard-coded and decided once and for all at the inception of the system. Roughly speaking, $k(D_{max})$ represents an upper bound on the number of blocks that the network creates in one unit of delay and that may not be referenced by one another. Section 4 discusses this parameter in more detail.

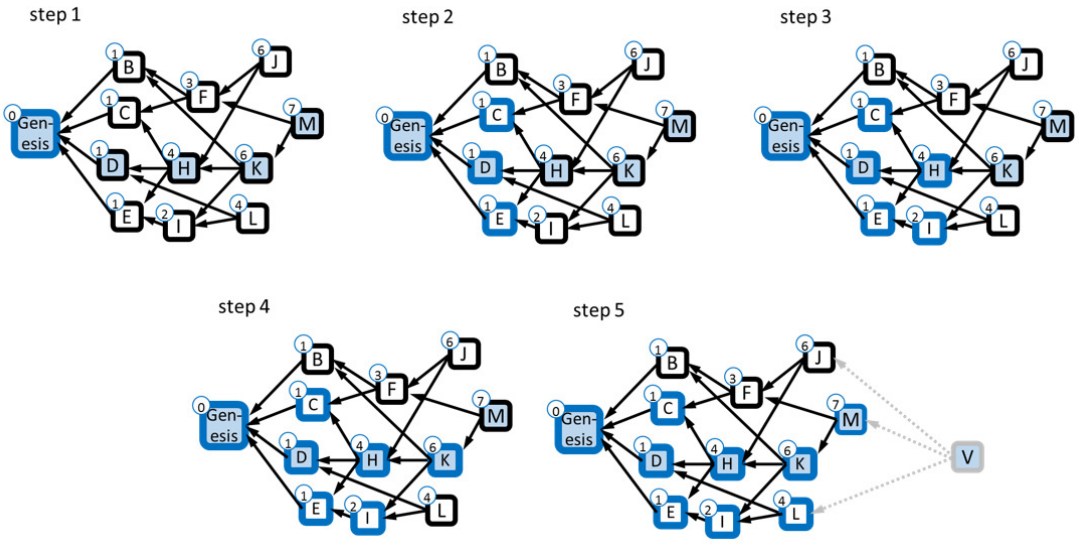

Figure 3: An example of a blockDAG G and the operation of GHOSTDAG to construct its blue set $BLUE_k(G)$ set, under the parameter $k = 3$. The small circle near each block X represents its score, namely, the number of blue blocks in the DAG $past(X)$. The algorithm selects the chain greedily, starting from the highest scoring tip M, then selecting its predecessor K (the highest scoring tip in $past(M)$), then H, D (breaking the C, D, E tie arbitrarily), and finally $Genesis$. For methodological reasons, we add to this chain a hypothetical "virtual" block V – a block whose past equals the entire current DAG. Blocks in the chain $(genesis, D, H, K, M, V)$ are marked with a light-blue shade. Using this chain, we construct the DAG's set of blue blocks, $BLUE_k(G)$. The set is constructed recursively, starting with an empty one, as follows: In step 1 we visit D and add $genesis$ to the blue set, as the only block in $past(D)$. Next, in step 2, we visit H and add to $BLUE_k(G)$ blocks that are blue in $past(H)$, namely, C, D, E. In step 3 we visit K and add H, I; note that block B is in $past(K)$ but was not added to the blue set, since it has 4 blue blocks in its anticone. In step 4 we visit M and add K to the blue set; again, note that $F \in past(M)$ could not be added to the blue set due to its large blue anticone. Finally, in step 5, we visit the block $virtual(G) = V$, and add M and to $BLUE_k(G)$, leaving L away due to its large blue anticone, and leaving J away because adding it would cause I to suffer too large a blue anticone (it already has C, D, and H in it).

3.2 Mining framework

Proof-of-work. Nodes create blocks of transactions by solving Proof-of-work puzzles. Block creation follows a Poisson process with parameter λ. For the sake of simplicity, we assume that λ is constant.[6] We denote by $time(B)$ the absolute time of B's creation. The computational power of node $v \in \mathcal{N}$ is captured by $0 < \alpha_v < 1$, which represents the probability that node v will be the creator of the next block in the system (at any point in time; this is a memoryless process).

Block references. Every block specifies its direct predecessors by referencing their ID in its header (a block's ID is obtained by applying a collision resistant hash to its header); the choice of predecessors will be described in the next subsection. This results in a structure of a direct acyclic graph (DAG) of blocks (as blocks can only reference blocks created before them), denoted typically $G = (\mathcal{V}, E)$. Here, $\mathcal{V}$ represents blocks and E represents the hash references. We will frequently write $B \in G$ instead of $B \in \mathcal{V}$.

DAG topology. The topology of the blockDAG induces a natural partial ordering over blocks, as follows: if there is a path in the DAG from block C to block B we write $B \in past(C)$; in this case, C was provably created after B and therefore B should precede C in the order.[7] A node does not consider a block as valid until it receives its entire past set. The unique block *genesis* is the block created at the inception of the system, and every valid block must have it in its past set.

Similarly, the future set of a block, $future(B)$, represents blocks that were provably created after it: $B \in past(C) \iff C \in future(B)$. In contrast to the past set, the future set of a block keeps growing in time, as more blocks are created and are referencing it. To avoid ambiguity, we write $future(B) \cap G$ or $future(B, G)$, and use $future(B)$ only when the context is clear or unimportant.

The set $anticone(B, G)$ represents all blocks in G that belong neither to B's future nor to its past (excluding B as well). These are blocks whose ordering with respect to B is not defined by the partial ordering that the topology of the DAG induces. Formally, for two distinct blocks $B, C \in G$: $C \in anticone(B, G) \iff (B \notin past(C) \wedge C \notin past(B)) \iff B \in anticone(C, G)$. Here too we usually specify the context, $anticone(B, G)$, because the anticone set can grow with time. In Figure 1 above we illustrate this terminology.

[6] In practice, λ must occasionally be readjusted to account for shifting network conditions. PHANTOM can support a retargeting mechanism similar to Bitcoin's, e.g., readjust every time that $Chn(G)$ grows by 2016 blocks.

[7] Note that an edge in the DAG points back in time, from the new block to previously created blocks which it references.

Yonatan Sompolinsky, Shai Wyborski, and Aviv Zohar

DAG mining protocol. G_t^v denotes the block DAG that node $v \in \mathcal{N}$ observes at time t. This DAG represents the history of all (valid) block-messages received by the node.

A *tip* of the DAG is a leaf-block, namely, a block with in-degree 0. The instructions to a miner in the DAG paradigm are simple:

(1) *When creating or receiving a block, transmit it to all of one's peers in $\mathcal{N}$. Formally, this implies that $\forall v, u \in honest : G_t^v \subseteq G_{t+D}^u$.*
(2) *When creating a block, embed in its header a list containing the hash of all tips in the locally-observed DAG. Formally, this implies that if block B was created at time t, by honest node v, then $past(B) = G_t^v$.*[8]

Since these are the only two mining rules in our system, a byzantine behaviour of the attacker (which controls up to α of the mining power) amounts to an arbitrary deviation from one or both of these instructions.

For convenience, we additionally regard the virtual block of the DAG, $virtual(G)$, which is a hypothetical (un-mined) block which points to the DAG's tips as its parents. Thus, $past(virtual(G)) = G$. Essentially, $virtual(G)$ represent the block template for the next block to be created by the miner, if it is honest.

3.3 DAG client protocol

The DAG as described so far possibly embeds conflicting transactions. In our work, these conflicts are resolved via an ordering protocol, namely, a protocol through which all nodes agree on the order of all transactions in the system. We refer to the ordering rule as *the client protocol*, as every participant in the network can run the ordering procedure on its client locally without any need to communicate additional messages with other clients.

Once such an order is agreed, one can iterate over all transactions in the prescribed order and approve each transaction that is *consistent* with those approved so far. In the case where transactions represent payments in Bitcoin's UTXO model, a transaction would be consistent with the set of previously approved transactions if its inputs belong to this set and none of these inputs are already spent by another transaction in the set (i.e., no "double spend"). However, our work is agnostic to the precise definitions of the transaction space and of the underlying consistency notion—for our purpose, suffice it to regard an abstract transaction space $\mathcal{U}$ and an abstract consistency function that determines whether a set of transactions $T \subseteq \mathcal{U}$ is consistent or not.

Formally, an ordering rule *ord* takes as input a blockDAG G and outputs a linear order over G's blocks, $ord(G) = (B_0, B_1, \ldots, B_{|G|})$. Transactions in the same block are ordered according to their appearance in it, and this convention allows us to talk henceforth on the order of blocks only. With respect to a given rule *ord*, we write $B \prec_{ord(G)} C$ if the index of B precedes that of C in $ord(G)$; we abbreviate and write $B \prec_G C$ or even $B \prec C$ when the context is understood. For convenience, we use the same notation $B \prec_G C$ when $B \in G$ but $C \notin G$.

[8] Technically it is more accurate to write $past(B) = G_t^v \setminus \{B\}$, as a block does not belongs to its own past set.

3.4 Convergence of the order

The following definition captures the desired security of the protocol, in terms of the probability that some order between two blocks will be reversed.

Definition 2. *Fix a rule ord and a node $u \in \mathcal{N}$. Let $B \in G_t^u$. The function $Risk_u$ is defined by the probability that from the point of view of u a block that did not precede B in time $t + r$ will later come to precede it:*

$$Risk_u(B, t, r) := \Pr\left(\exists s > t + r, \exists C \in G_s^u : B \prec_{G_{t+r}^u} C \wedge C \prec_{G_s^u} B\right)$$

If $B \notin G_t^u$ we define $Risk_u(B, t, r) = 0$.

The function Risk is the maximal $Risk_u$ over honest nodes:

$$Risk(B, t, r) = \max_{u \in honest} Risk_u(B, t, r)$$

In the definition above, the probability is taken over all random events in the network, including block creation and propagation, as well as the attacker's arbitrary (byzantine) behaviour. The convergence property below guarantees that the order between a block and those succeeding it (or those not published yet) will not be reversed, *w.h.p.* This captures the security of the protocol, as it provides honest nodes with (probabilistic) security guarantees regarding possible reorgs.

Property 1. *An ordering rule ord is said to (1-α)-*converge *if $\forall t > 0$ and B: $\lim_{r \to \infty} Risk(B, t, r) = 0$, even when a fraction α of the mining power is byzantine.*

Ideally, we would want $Risk(B, t, r) = 0$ to vanish exponentially fast so as to ensure speedy confirmation times.

Remark. *Property 1 essentially couples the Safety and Liveness properties required from consensus protocols. Indeed, once $Risk(B, t, r) < \epsilon$, a decision to accept transactions in B can be made (Liveness), and is guaranteed to be irreversible (Safety) up to an error probability of ϵ—as in Nakamoto Consensus and similar protocols, decisions are only irreversible in the probabilistic sense. Nevertheless, we avoid phrasing our results in these terms, for the sake of clarity of presentation. The complication arises from the need to analyze the system from the perspective of* every *node G_t^v; this technicality is not unique to PHANTOM, and should be regarded in any work that formalizes blockchain based consensus (unless propagation delays are assumed to be negligible). We leave the task of bridging this gap to a later version.*

The security threshold is the minimal portion of the hashing power that an attacker must acquire in order to disrupt the protocol's operation:

Definition 3. *The* security threshold *of an ordering rule ord is defined as the maximal α (attacker's relative computational power) for which ord (1-α)-*converges *exponentially fast.*

3.5 Main result

A protocol is scalable if it is safe to increase the block creation rate λ without compromising the security, that is, if the security threshold does not deteriorate as λ increases (this can be phrased also in terms of increasing the block size b rather than λ).

We claim that GHOSTDAG is indeed a scalable protocol, in this sense:

Theorem 4 (GHOSTDAG scales). *Given a block creation rate $\lambda > 0$, $\delta > 0$, and $D_{max} > 0$, if D_{max} is equal to or greater than the network's propagation delay diameter D, then the security threshold of GHOSTDAG, parameterized with $k(D_{max}, \delta)$, is lower bounded by $\frac{1}{2} \cdot (1 - \delta)$.*

The parameterization of GHOSTDAG via $k(D_{max}, \delta)$ is defined in the subsequent section. Theorem 4 encapsulates the main achievement of our work. We prove the theorem formally in Section A. Contrast this result to a theorem regarding Nakamoto Consensus, which appears in several forms in previous work (e.g., [11, 15]):

Theorem 5 (Nakamoto Consensus does not scale). *The security threshold of Nakamoto Consensus goes to 0 as $D \cdot \lambda$ grows.*

Finally, we note that even if $D_{max} \not\geq D$, the system's security does not immediately break apart. Rather, the minimal power needed to attack the system goes from 50% (times $(1 - \delta)$) to 0, deteriorating at a rate that depends on the error gap $D - D_{max}$.

4 SCALABILITY AND NETWORK DELAYS

4.1 The propagation delay parameter D_{max}

The scalability of a distributed algorithm is closely tied to the assumptions it makes on the underlying network, and specifically on its propagation delay $D = D(b)$ where b is the block-size in KB. The real value of D is both unknown and sensitive to shifting network conditions. For this reason, Bitcoin operates under the assumption that D is much smaller than 10 minutes, and sets the average block interval time to 10 minutes. While this seems like an overestimation of the network's propagation delay under normal conditions (at least in 2018's Internet terms), some safety margin must be taken, to account for peculiar network conditions as well. Similarly, in PHANTOM (and GHOSTDAG) we assume that the unknown D is upper bounded by some $D_{max} = D_{max}(b)$ which is known to the protocol. The protocol does not explicitly encode D_{max}, rather, it is used to parameterize k, as will be described in the next subsection.

The use of an *a priori* known bound D_{max} distinguishes GHOSTDAG's (and PHANTOM's) security model from that of SPECTRE [14]. While the security of both protocols depends on the assumption that the network's propagation delay D is upper bounded by some constant, in SPECTRE the value of such a constant need not be known or assumed by the protocol, whereas GHOSTDAG makes explicit use of this parameter (via k) when ordering the DAG's blocks. The fact that the order between any two blocks becomes robust in GHOSTDAG, but not in SPECTRE, should be ascribed to this added assumption; see further discussion in Section 5.

4.2 The anticone size parameter k

The parameter k is decided from the outset and hard-coded in the protocol. It is defined as follows:

$$k(D_{max}, \delta) := \min\left\{\hat{k} \in \mathbb{N} : f(\hat{k}, D_{max}) < \delta\right\} \tag{1}$$

$$f(\hat{k}, D_{max}) := \max\left\{\sum_{j=\hat{k}+1}^{\infty} e^{-2\cdot C} \cdot \frac{(2\cdot C)^j}{j!}, \frac{2\cdot C}{\hat{k}+2\cdot C}\right\}, \tag{2}$$

where $C := D_{max} \cdot \lambda$. The motivation behind this definition is twofold.

Natural anticone size. First, we want to devise a bound over the number of blocks created in parallel. Since the block creation rate follows a Poisson process, for an arbitrary block B created at time t, $k(D_{max}, \delta)$ bounds the number of additional blocks created in the time interval $[t - D_{max}, t + D_{max}]$, with probability of at least $1 - \delta$; the term $\sum_{j=\hat{k}+1}^{\infty} e^{-2\cdot D_{max}\cdot\lambda} \cdot \frac{(2\cdot D_{max}\cdot\lambda)^j}{j!}$ from (2) bounds the probability that more than k blocks were created in parallel to B in the time interval $[t - D_{max}, t + D_{max}]$.

Observe that blocks created in the intervals $[0, t - D_{max})$ and $(t + D_{max}, \infty)$, by honest nodes, belong to B's past and future sets, respectively. Consequently, in principle, $|anticone(B)| \leq k$ with probability of $1 - \delta$ at least. However, an attacker can artificially increase B's anticone by creating blocks that do not reference it and by withholding his blocks so that B cannot reference them.

Chain growth rate. The second motivation for (1) is potential manipulations by attackers on the increase rate of the blue set. GHOSTDAG does not guarantee that discovering new blocks necessarily increases the size of the blue set. Fortunately, by increasing k, we are able guarantee that the adverse effect is arbitrarily small. Indeed, A worst case analysis carried out in Lemma 9 shows that the overall decrease is at most by a factor of $\left(1 - \frac{2\cdot C}{k+2\cdot C}\right)$. This is the role of the right-hand size term inside (2).

We note one caveat here: the probability $\frac{2\cdot C}{k+2\cdot C}$ does not vanish exponentially fast with k. To fix this, a tighter analysis of the effect of these attacks is needed; we leave this challenge to future work.

4.3 Trade-offs

Theorem 4, and the parameterization of GHOSTDAG in (1), tie together k, D_{max}, λ, and δ. Striving for a better performance by modifying one parameter (e.g., increasing λ to obtain larger throughput and more frequent blocks) must be understood and considered against the effect on all other parameters.

Increased block creation rate. Although the security threshold does not deteriorate as λ is increased, λ cannot be increased indefinitely, or otherwise the network becomes congested. The value of λ should be set such that nodes that are expected to participate in the system can support such a throughput. For instance, if nodes are required to maintain a bandwidth of at least 1 *MB* per second, and blocks are of size $b = 1$ MB, then the block creation rate should be set to $\lambda = 1$ blocks per second (this is merely a back-of-the-envelope calculation, and in practice other messages consume the bandwidth as well).

Higher security threshold. Theorem 4 states the security threshold in terms of δ. Following (1) we notice that tightening the

Yonatan Sompolinsky, Shai Wyborski, and Aviv Zohar

security threshold – by choosing a lower δ – requires increasing k. A large k leads to slow confirmation times, as will be discussed shortly.[9]

Larger safety margin. Similarly, if D_{max} is to be increased, one needs to increase k as well in order to maintain the same security level (represented by δ).

As discussed in Subsection 4.1, it is better to overestimate D and choose a large D_{max} in order to remain on the safe side.[10] Recall that the security of Bitcoin's chain depends on the assumption that $D \cdot \lambda \ll 1$, namely, that w.h.p. at least D seconds pass between consecutive blocks, so that forks are rare. Thus, Bitcoin's large safety margin over D suppresses its throughput severely as it requires selecting a very low block rate $\lambda = 1/600$ (one block per 10 minutes). This is not the case with GHOSTDAG's DAG, as the security of the DAG ordering does not rely on the assumption $D \cdot \lambda \ll 1$. Therefore, even if we overestimate D, we can still allow for very high block creation rates while maintaining the same level of security. Consequently, GHOSTDAG supports a very large throughput, and does not suffer from a security-scalability tradeoff.

That said, in GHOSTDAG there is still a tradeoff between a large safety margin and fast convergence of the protocol. A gross overestimation of D_{max} – resulting an increase in k – would significantly increase the waiting time for transaction settlement. Thus, D_{max} should be set to a reasonable level. In Section 5 we discuss how this tradeoff can be restricted to visible conflicts only, and how applications such as payments can enjoy much faster confirmation times.

A note on the effective throughput. To increase the number of transactions included in the DAG per second, the protocol designer can increase the block rate λ or/and the block size b (which in turn affects $D_{max} = D_{max}(b)$). However, if honest miners include in their blocks the very same transactions, this would not contribute to the number of *unique* transactions included in the DAG, per second. There are two approaches to fix this. The first is to observe that honest and rational miners are actually incentivized to randomize over the transactions available to them in order to avoid "collision" and to thereby maximize their own profit; see more on this in [8]. The second approach is to shard the transaction space and allow the block to contain transactions from one shard only, in a way that guarantees that parallel blocks will not collide on the same shard, with high probability. This technique was used in some of the works mentioned in Section 7, under "Parallel Nakamoto Chains". Both techniques can be deployed in GHOSTDAG. Consequently, increasing the block rate or block size does indeed lead to a linear increase in the effective throughput.

5 CONFIRMATION TIMES

As in Nakamoto Consensus, the waiting time for transaction confirmation depends on the assumed attacker size α, and on the allowed error ϵ. The security analysis given in Appendix A shows that within constant expected time the chain of the honest network enters into a block race with any hypothetical or actual attacker chain. It is implied that the waiting time for the ordering between a given block B and other blocks becomes robust at a rate of $O\left(\log_{\frac{\alpha}{1-\alpha}}(1/\epsilon)\right)$. This analysis was asymptotic, and we leave the task of tightening the analysis and improving the constants (hidden in the O) for future work.

Still, we observe that in the case of payments, transactions of honest users can be confirmed much faster. Indeed, an honest user will not publish a conflicting transaction, and her transaction will therefore be commutative with all other published transactions. Of course, the payee does not know *a priori* who of the payers is honest, and will thus wait until the block containing the transaction is guaranteed (w.h.p.) to precede any new block that might be published by the attacker. Blocks that are published in the interim will not contain a conflicting transaction, in the case of an honest payer, and will therefore not delay acceptance.

Formally, we argue that GHOSTDAG enjoys the following property: Given a published block b, the probability that a newly added block in $anticone(b)$ will be accepted as a blue block decays exponentially:

Proposition 6. *If blocks b and c were published at times t and $t+r$, respectively, and $c \in anticone(b)$, then the probability that c will ever be considered blue is $O(e^{-C \cdot r})$ for some positive constant C.*

Proof. Lemma 9 implies that the number of adversary blocks that are in the blue set of an honest node and not in $past(B)$ grows linearly with r. Let $\mathcal{B}_r = future(B) \cap past(B_r)$ where B_r is the honest tip of the node just before r. Then for any N and for any sufficiently large r it holds that $|\mathcal{B}_r| \leq N + k$. All blocks in $\mathcal{B}_r$ are in $anticone(C)$, for any $C \in anticone(B)$ that was discovered after $t + r$: indeed, if $B' \in \mathcal{B}_r$ satisfies $B' < C$ then $B' < B$, and if $C < B'$ then it was discovered before $t + r$. Therefore, if B_r remains the honest tip at time $t + r$, then C is not in the blue set of the honest node, as it has at least $|\mathcal{B}_r| > k$ blue blocks in its anticone.

Let B' be a block such that $C \in BlueSet(B')$, then $|BlueSet(B') \cap \mathcal{B}_r| \leq k$, or otherwise C cannot be inside $BlueSet(B')$. This proves that for the honest node to consider C blue, a block B' has to win a block-race with B_r – up to $3k$ blocks which it can freeload – which started a time t the latest and lasted up to $t + r$ at least. The probability of winning such a block-race is in $O(e^{-C \cdot r})$. □

In fact, following a similar analysis to that developed in [13, 14], the probability of C becoming blue is

$$\sum_{m=0}^{\infty} \binom{m+N+k-1}{m} \cdot \alpha^m \cdot (1-\alpha')^{N+k} \cdot \left(\frac{\alpha}{1-\alpha'}\right)^{\max(N-m,0)}$$

where $1 - \alpha' = (1 - \alpha) \cdot (1 - \delta)$ as derived in Lemma 9. This implies, practically, that a merchant who received a payment in a block B, listens to the blockDAG and sees no conflicting payment in any block in B's anticone, can accept the payment when the latter term becomes smaller than ϵ (- the error probability allowed by the merchant).

For example, if the block creation rate is set to $\lambda = 1$ block per second, and the assumed upper bound on the network's round trip time ($2 \cdot D$) is 7 seconds, then k can safely be set to 16. Assuming an attacker with $\alpha \leq 0.25$, and an allowed error of $\epsilon = 0.1\%$ (the values of these parameters must be assumed in Nakamoto Consensus as

[9] The advanced reader should notice that although increasing λ has a similar negative effect on k, it has at the same time a positive effect on confirmation times, and so a certain λ will be optimal as far as confirmation times are concerned.

[10] Several blockchain based projects do not do so, and consequently compromise the security threshold of their system.

well), the waiting time for transaction confirmation would be in the order of 45 seconds. Note that if the actual network delay is smaller than the 7 seconds bound – say, below 4 seconds, as in Bitcoin's Relay Network [1] – then the waiting time will remain the same. This demonstrates that GHOSTDAG is not responsive to the actual network delay. For comparison, under the same conditions, SPECTRE's waiting time would be in the order of 21 seconds. This is due to SPECTRE not assuming any upper bound within the protocol, which enables it to be responsive.

6 VARIANTS

In Section 2 we described PHANTOM's greedy algorithm to mark blocks as blue or red (Algorithm 1). In fact, similar algorithms may be able to provide similar guarantees. We describe below two such greedy variants, which We *conjecture* that Algorithm 1 can be replaced with each of these variants, and perhaps even improve over its performance as analyzed in Section 5.

6.1 Iterative elimination of blocks

The first proposed variant is based on an iterative method common in combinatorial optimization. The algorithm works as follows: Given a block DAG G, we iteratively eliminate from the Blue set the block with largest Blue anticone, and continue doing so until we arrive at a k-cluster:

Algorithm 2 Selection of a blue set

Input: $G = (V, E)$ – a block DAG, k – the propagation parameter
Output: $BLUE_k(G)$ – the dense-set of G

function CALC-BLUE(G, k)
  $BLUE_k(G) \leftarrow V$
  while $\exists b \in BLUE_k(G)$ with $|anticone(b) \cap BLUE_k(G)| > k$ **do**
    $c \leftarrow \arg_b \max \{|anticone(b) \cap BLUE_k(G)|\}$ (with arbitrary tie-breaking)
    remove c from $BLUE_k(G)$
  return $BLUE_k(G)$

This variant can be thought of, *informally*, as a greedy approximation to the Maximum k-cluster SubDAG problem described in Section 1. In order to understand the intuition behind this algorithm consider a straw-man algorithm for finding a large enough k-cluster: *Remove all blocks with anticone larger than k*. Such a method would clearly fail since the attacker can artificially increase the anticone of all honest blocks by publishing a chain that is disconnected from the entire DAG (apart from the genesis). Instead, Algorithm 2 first penalizes and removes (from the blue set) blocks that suffer from the largest anticones, then updates the (blue) anticone sizes and checks whether there still exist blocks with too large of anticones. It is clear how publishing a disconnected chain will fail, with this iterative elimination method, and we leave the analysis of its security against arbitrary attack methods to future work.

6.2 SPECTRE as a colouring method for PHANTOM

SPECTRE is a protocol that outputs a pairwise ordering over DAGs [14]. Given a DAG G, and two blocks b, c in G, let us write $b \prec_G c$ if b precedes c in the ordering outputted by SPECTRE. The gap of a block is defined by the number of blocks in its anticone that precede it in the order: $gap(b, G) := |\{c : c \prec_G b\}|$. Similarly to the size of a block's anticone, an honest block regularly enjoys a small gap, and outlier blocks with too large a gap can be suspected as suspicious and malicious; however, as in the previous subsection, a sophisticated attacker may be able to artificially increase the gap of many honest blocks simultaneously. To that end, we focus on a more subtle metric, defined in the SPECTRE work: Given a block b in a DAG G, we are interested in b's "pumping" index, which informally measures by how much b's weight needs to be increased in order for b to precede all other blocks in its anticone. Formally, for any $j \in \mathbb{N}$, $\langle b, G, j\rangle$ represents the DAG G modified as follows: j blocks $b_1, \ldots, b_j$ are added to the immediate future of b such that all of b's immediate children in G are immediate children of b_j in $\langle b, G, j\rangle$, and b_1 is b's only immediate child in $\langle b, G, j\rangle$. Finally, *pumping_idx* is the minimal j such that b enjoys a gap of zero in $\langle b, G, j\rangle$: $pumping\_idx(b, G) := \min\{j : gap(b, \langle b, G, j\rangle) = 0\}$.[11]

The pumping index $pumping\_idx(b, G)$ can be considered as b's pumping index, and we conjecture that eliminating all blocks with $dist\_gap(b, G)$ equal to or smaller than k yields a secure algorithm which enjoys SPECTRE's speed. The method is summarized in the following method:

Algorithm 3 Selection of a blue set

Input: $G = (V, E)$ – a block DAG, k – the propagation parameter
Output: $BLUE_k(G)$ – the dense-set of G

function CALC-BLUE(G, k)
  $BLUE_k(G) \leftarrow \emptyset$
  $\prec_G \leftarrow SPECTRE(G)$
  for $b \in G$ **do**
    if $pumping\_idx(b, G) \leq k$ **then**
      add b to $BLUE_k(G)$
  return $BLUE_k(G)$

As mentioned, we conjecture that this method too leads to a secure protocol (in terms of Theorem 4). Implementing Algorithm 3 may be impractical, and it is therefore more interesting on a theoretical basis. Indeed, since SPECTRE is responsive and converges at the rate of the actual network delay D (rather than the rate controlled by the protocol's upper bound over D), Algorithm 3 is likely to produce faster confirmation times than GHOSTDAG. Analysis of this algorithm is left for future work as well.

7 RELATED WORK

Many suggestions to improve the scalability of permissionless blockchains have been proposed in recent years. These proposals fall into two main categories, *on-chain scaling* and *off-chain scaling*.

On-chain scaling. The protocols in this category may differ, e.g., in how fast blocks are created, how blocks are organized in the

[11] Note that, in SPECTRE, the pumping index was denoted $distdist$.

Yonatan Sompolinsky, Shai Wyborski, and Aviv Zohar

ledger (a chain, a tree, a DAG, etc.), which transactions in the ledger are considered valid, and more. PHANTOM belongs to this line of works. Previous works in this family of protocols includes GHOST [15], where a main chain of blocks is chosen according to a greedy algorithm and not through the longest chain rule; Inclusive [8], where any chain-selection rule is extended to an ordered DAG and transactions off the main chain are added in a consistent manner; Bitcoin NG [3], where the ledger consists of slow *key blocks* (containing no transactions) and fast *microblocks* that contain transactions. The sole purpose of key blocks in Bitcoin NG is to define the miner that is eligible to create microblocks in that epoch and confirm thus transactions at a high rate.

GHOST is still susceptible to some attacks, one of which was described in [7]. The DAG in Inclusive adds throughput but not security to the main chain, hence suffers from the same limitations as the underlying main chain selection rule. Key blocks in Bitcoin NG are still generated slowly, thus confirmation times remain high.

Our work is most similar to the SPECTRE protocol [14]. SPECTRE enjoys both high throughput and fast confirmation times. It uses the structure of the DAG as representing an abstract vote regarding the order between each pair of blocks. One caveat of SPECTRE is that the output of this pairwise ordering may not be extendable to a total order, due to possible Condorcet cycles. Another related caveat is that SPECTRE does not guarantee convergence of the ordering between two blocks that were published in time proximity to one another. This *weak* liveness property is showed to suffice for the use case of payments, where conflicts in two such blocks can only harm a malicious user that published a double spend. PHANTOM solves these issue and provides a linear ordering over the blocks of the DAG. As such, PHANTOM can support consensus regarding any general computation, including general *Smart Contracts*, which SPECTRE cannot handle. Indeed, in order for a computation or contract to be processed correctly and consistently, the full order of events in the ledger is usually required, and particularly the order of inputs to the contract.[12] PHANTOM's linear ordering does not come without cost—confirmation times are mush slower than those in SPECTRE. In Section 5 we discuss the confirmation times in PHANTOM compared to those of SPECTRE, when the user does not publish a visible double spend.

Parallel Nakamoto Chains. Another line of work attempts to avoid the scalability-security tradeoff imposed by the longest chain rule by dividing the ledger into k separate Nakamoto chains [2, 4, 16]. The basic technique behind these protocols is to assign a block to one of the chains only after it was successfully mined, using the proof-of-work randomness represented in (the hash of) its header. This randomness ensures that an attacker cannot choose which chain to concentrate its attack on. One way to view these protocols and compare them against PHANTOM (and GHOSTDAG) is that Nakamoto Consensus operates under the assumption that $D \cdot \lambda \ll 1$, PHANTOM under the assumption that $D \cdot \lambda \ll k$, and these parallel chain protocols under $D \cdot \lambda / k \ll 1$; indeed, by dividing the protocol into k separate chains, each chain is mined at a rate of λ / k, and this inequality ensures that each chain enjoys a negligible orphan rate similarly to Nakamoto Consensus.

Protocols in this paradigm must specify the order over blocks in the ledger of parallel chains. The performance of these protocols depends on their respective ordering rules, and we differ the task of rigorously comparing them to GHOSTDAG to the full version of this paper.

An attack on a previous variant. The work in [9] presents a DAG-based protocol Conflux, which is identical to Inclusive-GHOST [8, 15]. In the appendix section of [9] there appears an attack on a previous version of PHANTOM. In that version, the greedy algorithm inherited the blue set S of the maximal tip, and added blocks to S as long as their anticone in S was of size k or less. The fact that the greedy algorithm – in contrast to vanilla PHANTOM– did not attempt at enforcing the k-cluster property on the chosen set of blue blocks was shown in [9] to be exploitable by an attacker. The greedy algorithm presented in this updated version of the paper, GHOSTDAG, does enforce the k-cluster property; we utilize this fact to prove its correctness. We thank the authors of [9] for highlighting this subtlety.

Off-chain scaling. Our work is orthogonal and complementary to off-chain scaling solutions, and can enhance their operation by orders-of-magnitude. For instance, when the DAG is used to serve channel-settlement transactions of Bitcoin's Lightning Network [12], it allows for a much cheaper access (due to larger supply of blocks and capacity) and much faster processing than if the LN were operating over a chain.

8 DISCUSSION

In this work we have introduced the PHANTOM paradigm, which generalizes over Nakamoto's chain into a DAG, and which imposes no *a priori* constraint over the system's throughput, thus avoiding the scalability-security tradeoff imposed by Nakamoto Consensus. We described a greedy algorithm, GHOSTDAG, which is more practical to implement, and proved its security rigorously. There are still several open research questions regarding GHOSTDAG: a tight analysis of its confirmation times, a comparison between it and the variants suggested in Section 6, and a security analysis of the original PHANTOM protocol which wasn't provided in this work.

[12] Contracts that do not require such a strict ordering can indeed be served under SPECTRE as well.

REFERENCES

[1] Bitcoin's relay network stats. http://bitcoinfibre.org/stats.html. Accessed: 2020-01-20.

[2] Vivek Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. Prism: Deconstructing the blockchain to approach physical limits. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 585–602, 2019.

[3] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. Bitcoin-ng: A scalable blockchain protocol. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 45–59, 2016.

[4] Matthias Fitzi, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Parallel chains: Improving throughput and latency of blockchain protocols via parallel composition. *IACR Cryptology ePrint Archive*, 2018:1119, 2018.

[5] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 281–310. Springer, 2015.

[6] Michael R Garey and David S Johnson. *Computers and intractability*, volume 29. wh freeman New York, 2002.

[7] Aggelos Kiayias and Giorgos Panagiotakos. On trees, chains and fast transactions in the blockchain. Cryptology ePrint Archive, Report 2016/545, 2016.
[8] Yoad Lewenberg, Yonatan Sompolinsky, and Aviv Zohar. Inclusive block chain protocols. In *International Conference on Financial Cryptography and Data Security*, pages 528–547. Springer, 2015.
[9] Chenxing Li, Peilun Li, Dong Zhou, Wei Xu, Fan Long, and Andrew Yao. Scaling nakamoto consensus to thousands of transactions per second. *arXiv preprint arXiv:1805.03870*, 2018.
[10] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. *IACR Cryptology ePrint Archive*, 2016:454, 2016.
[11] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model, 2016.
[12] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. *Technical Report (draft)*, 2015.
[13] Meni Rosenfeld. Analysis of hashrate-based double spending. *arXiv preprint arXiv:1402.2009*, 2014.
[14] Yonatan Sompolinsky, Yoad Lewenberg, and Aviv Zohar. Spectre: A fast and scalable cryptocurrency protocol. *IACR Cryptology ePrint Archive*, 2016:1159, 2016.
[15] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 507–527. Springer, 2015.
[16] Haifeng Yu, Ivica Nikolic, Ruomu Hou, and Prateek Saxena. Ohie: Blockchain scaling made simple. *arXiv preprint arXiv:1811.12628*, 2018.

A GHOSTDAG SCALABILITY PROOF

We present a prove that the GHOSTDAG order converges, and that an attacker (with less than 50%) is unable to cause reorgs. We restate the theorem from Section 3:

Theorem 4 (GHOSTDAG scales) *Given a block creation rate $\lambda > 0$, $\delta > 0$, and $D_{max} > 0$, if D_{max} is equal to or greater than the network's propagation delay diameter D, then the security threshold of GHOSTDAG, parameterized with $k(D_{max}, \delta)$, is at least $\frac{1}{2} \cdot (1 - \delta)$.*

A.1 Assumptions and notations

We first remind the reader that the topology of the DAG is regarded in our context in reverse to the usual convention, namely, if there is a path from a block B to a block C we say that C is a *parent* of B, whereas most literature concerning DAGs would say that B is a parent (or predecessor) of C. In our context blocks always point to *older* blocks, so this anti-convention is biologically justified whereas the conventional terminology is misleading.

We assume a worst case byzantine attacker which suffers no internal delays or delays from or to honest nodes, and which deviates from the mining protocol arbitrarily (see definition). Still, regardless of any attacker intervention in the network, he cannot cause messages between honest nodes to take longer than D_{max} to propagate. In the sequel, to avoid clutter, we use D instead of D_{max} to denote the bound on the network delay (note that in previous sections D was used to denote the *actual* delay).

When discussing the honest network, we fix an arbitrary honest node $u \in honest$ and assume its point of view, whereby terms such as *the* honest chain should be interpreted from u's point of view. To that effect, by the *honest (attacker) tips* at time t we mean all immediate parents of the virtual block of the honest (attacker) node at time t, by the *honest (attacker) chain* we mean the selected chain of the virtual block, and by the *honest (attacker) blue set* we mean the blue set of the virtual block of the honest (attacker) node.

The attacker does not follow the consensus rules and in particular may create a block which points at blocks which are not tips, and withhold any blocks they create. Hence, we separate between the time a block was *created* (by the attacker) and by the time it was *discovered* (by the arbitrary honest node u).

The *honest score* $w_H(t)$ is defined as the score of the virtual block of the honest node at time t. The *advantage* of a block B is defined as $adv(B) = score(B) - w_H(t)$, where t is the time B was created. The *attacker advantage* $adv(t)$ is the maximum of $adv(B)$ over all attacker tips at time t. For a given block B, we define the *relative advantage* $adv_B(t)$ to be the maximum of $adv(B)$ over all attacker tips at time t such that B is *not* in their selected chain (or $-\infty$ if there are none). We say that *B convinced the honest node at time t* if it was discovered before $t - 2D$ and was in the selected chain at time t.

We use $B \to C$ to denote that C is an immediate parent of B and $B \rightrightarrows C$ to indicate that it is a selected parent. We use $C < B$ to denote that $C \in past(B)$ and $C <^\star B$ to denote at C is in the selected chain of B. We use $C \leq B$ and $C \leq^\star B$ to denote the same, with the possibility that $B = C$.

We define many constants throughout the proof with the proviso that they only depend "on system parameters", by which we mean that they only depend on λ, D and k (and not on t, on the particular DAG structure of any node, etc.).

A.2 Proof Structure

Due to the definition of security threshold in Definition 3, Theorem 4 reduces to the following proposition.

Proposition 7 (GHOSTDAG security)**.** *For any t, r, the probability that the ordering of two blocks published before time t will change after time $t + r$ is $O(e^{-cr})$, where c depends only on the system parameters.*

In terms of Section 3, Proposition 7 states that $Risk_u(B, t, r)$ decays exponentially fast, at a rate which does not depend on u or t. This implies that $Risk(B, t, r)$ decays at the same rate, and in particular that the GHOSTDAG ordering rule $(1 - \alpha)$-converges.

The correctness of this Proposition 7 follows from the following proposition:

Proposition 8. *Assume that B convinced the network at time t, then $adv_B(t)$ is bound from above by a Markovian process on $\mathbb{Z}$ which goes to $-\infty$ exponentially fast.*[13].

The following two lemmas, which are of independent interest, are part of the proof of Proposition 8. The first lemma states that the growth rate of the blue score of the public chain is $1 - \alpha$, up to a factor arbitrarily close to 1:

Lemma 9. *The expected value of $w_H(t + r) - w_H(t)$ is at least $(1 - \alpha)(1 - \delta) r\lambda$.*

The next lemma states that the attacker's advantage can be capped with high probability:

Lemma 10. *The advantage $adv(t)$ is upper bounded by a stochastic process which admits a stationary distribution with an exponentially decaying tail.*

[13] In the same sense that a random walk on $\mathbb{Z}$ which is biased to the left goes to $-\infty$ exponentially fast. That is, if T is the transition operator and p is a finitely supported distribution on $\mathbb{Z}$, then for any $N \in \mathbb{Z}$ it holds that $\|\Pi_{[N,\infty)} T^{O(N)} p\|_1 = O(c^N)$ for some constant $c < 1$ (which is $\frac{\alpha}{1-\alpha}$ for a random walk which transitions right with probability $\alpha < 1/2$)

Yonatan Sompolinsky, Shai Wyborski, and Aviv Zohar

A.3 Blue set growth

In this section we discuss the non monotonicity of the chain weight and prove Lemma 9.

In Nakamoto Consensus the longest chain is chosen, and it is therefore impossible for an adversary to reduce the length of the longest chain by publishing blocks. Stated otherwise, the score of a Bitcoin node increases monotonically. In GHOSTDAG this no longer holds. Indeed, there are cases where by learning of new blocks, the blue score of the virtual node actually *decreases*. This might have an adverse effect on the growth of the blue set, that is, of the score of the honest network. This adverse effect is bounded by a constant factor of $\left(1 - \frac{2D\lambda}{k+2D\lambda}\right)$ which is already built into δ in the definition of the security threshold (1). That this is the appropriate factor is exactly the statement of Lemma 9, which we now prove.

Lemma 9. *The expected value of $w_H(t+r) - w_H(t)$ is at least $(1-\alpha)(1-\delta)r\lambda$.*

Proof. Assume that a block B was published that caused w_H to decrease. Let B' be the honest selected tip right before B was published. We must have that $B' \not< B$ and that $score(B) \geq score(B')$, then the reduction in score is at most the amount of blocks in $anticone(B')$ that the virtual node considered blue before B was discovered, and red after B was discovered. In particular, any such block must have an anticone larger than k. If all blocks are honest then the probability of this is at most δ. This proves the theorem for the special case of an attacker which indefinitely withholds their blocks.

To generalize to the adversarial setting, we need to argue the most damage an attacker can cause to the blue score by withholding blocks. Assume that at time t a block C was discovered that reduced the blue score, and let B be the honest selected tip right before C was published. If $B <^\star C$ then $score(C) > score(B)$, contradicting the hypothesis. It follows that $C \to B$ can only increase $score(C)$, so we may assume without loss that $C \not\to B$. It follows that $score(C) \geq score(B)$, for else B would have remained the honest selected tip and the score could not have decreased.

Let n_B be the number of blue blocks in the anticone of B that were in the honest blue when B was the selected tip, and let n_C be defined similarly for C, then the decrease in score is at most $n_B - n_C$. However, note that if the selected chain of C contains less than k blocks above $\max(SelectedChain(B) \cap SelectedChain(C))$ then $n_B \geq n_C$ (as any block in $anticone(B)$ could be a blue block, up to k blocks), and the score actually did not decrease.

It follows that the adversary gains the most by publishing $k+1$ blocks in the anticone of the selected parent such that the most recent published block has score at least as large as that of the selected parent. This will cause the honest network to switch a chain, such that all the blocks in the anticone of the old selected parent, except the $k+1$ blocks published by the adversary, will be considered red (as they are all in the anticone of the published $k+1$ blocks, which are blue). On average, there are at most $2D\lambda$ such blocks, so that the adversary has managed to replace $k + 2D\lambda$ blue blocks with $k+1$ blue blocks.

This implies that the highest factor by which the adversary can reduce the growth of the honest blue set is $\frac{k}{k+2D\lambda} = 1 - \frac{2D\lambda}{k+2D\lambda}$.

Let δ_0 be the probability that a particular honest block has an honest anticone larger than k. It follows that the expected value of $w_H(t) - w_H(s)$ is at least $(1-\alpha)(1-\delta_0)\left(1 - \frac{2D\lambda}{k+2D\lambda}\right)(t-s)\lambda > (1-\alpha)(1-\delta)(t-s)\lambda$, where the last inequality holds by equation 2 in the definition of k. □

A.4 Freeloading and Bounded Advantage

In this section we prove Lemma 10, by a reduction to the analysis carried in [14]. The main challenge in this reduction is that the attacker may use the work done by the honest network to boost the score of their competing chain, by including honest blocks in the blue set of the chain tip. We call this phenomenon *freeloading*, more formally:

Definition 11. *We say that $C \in \mathcal{A}$ is* freeloading off $B \in \mathcal{H}$ *if $B \in BlueSet(C)$, and $C' <^\star C$ implies $B \notin BlueSet(C')$.*

It turns out that the advantage an attacker can gain by freeloading is bound by a constant depending only on system parameters.

Lemma 12 (The Freeloader Bound)**.** *If C is freeloading off B then $score(C) \leq score(B) + 3k$.*

Proof. First note that if $C \rightrightarrows B$ then $score(C) \leq score(B) + k$, as GHOSTDAG does not add more than k blue blocks to C's blue set over that of its selected parent B. We subsequently treat the case where $C \not\rightrightarrows B$.

Let $D <^\star C$ be maximal such that $D < B$ (this set is not empty as it contains the genesis block).Then $score(B) > score(D)$. We show below that $score(C) - score(D) \leq 3k+1$ and conclude that $score(C) \leq score(B) + 3k$.

Note the following observations:

- $score(C) - score(D) \leq |BlueSet(C) \setminus past(D)|$,
- $BlueSet(C) \setminus past(D) \subset \mathcal{D}_1 \cup \mathcal{D}_2 \cup \mathcal{D}_3$ where: $\mathcal{D}_1 = past(C) \cap future(B)$, $\mathcal{D}_2 = past(B) \cap future(D)$, and $\mathcal{D}_3 = past(C) \cap anticone(B)$.
- $SelectedChain(C) \cap future(D) \neq \emptyset$: the contrary implies that $C \to D$. However, that $B < C$ implies that there is some $C \to B'$ such that $B \leq B'$. But $score(B') \geq score(B) > score(D)$, which implies $C \not\rightrightarrows D$, contradicting $D <^\star C$.

Now, let E_1, E_2 be the maximal and minimal elements of $SelectedChain(C) \cap future(D)$ respectively. If it does not hold that $C \rightrightarrows E_1$ then there must be a block between them, contradicting the maximality of E_1. Similarly, $E_2 \rightrightarrows D$.

If $\mathcal{D}_1$ intersects $past(E_1)$ then $B < E_1$, contradicting the minimality of C. Assume $D_1 \in \mathcal{D}_1 \cap future(E_1)$, then $score(E_1) < score(D_1)$ and since $D_1 < C$ we get that $C \not\rightrightarrows E_1$, which is a contradiction.

Similarly, if $\mathcal{D}_2$ intersects $future(E_2)$ then the maximality of D is contradicted, and if $\mathcal{D}_2$ intersects $past(E_2)$ then $E_2 \rightrightarrows D$ is contradicted.

Hence $\mathcal{D}_1 \subset anticone(E_1)$, $\mathcal{D}_2 \subset anticone(E_2)$ and $\mathcal{D}_3 \subset anticone(B)$ where it holds that $E_1, E_2, B \in BlueSet(c)$. Therefore $|\mathcal{D}_i \cap BlueSet(C)| \leq k$ for $i = 1, 2, 3$, and since $BlueSet(C) \cap (past(B) \cup \{B\}) \subset \mathcal{D}_1 \cup \mathcal{D}_2 \cup \mathcal{D}_3 \cup \{B\}$ it follows that $|BlueSet(C) \setminus past(D)| \leq 3k+1$. Hence $score(C) - score(D) \leq 3k+1$, whereby $score(C) \leq score(B) + 3k$.

□

Lemma 10 *The advantage $adv(t)$ is upper bounded by a stochastic process which admits a stationary distribution with an exponentially decaying tail.*

Proof. The work in [14] contains an analysis of the premining process, i.e., the maximal advantage of the attacker over an honest block, in a block race; formally: $adv'(t) := \max_{C\in[0,t]}\{future(C)\cap\mathcal{A} - future(C)\cap\mathcal{H}\}$ (the notation $adv'(t)$ does not appear therein). In contrast, we are interested in the process $adv(t) = w_A(t) - w_H(t)$, which is similarly bounded: $adv(t) = score(C) - score(V)$ where C is the selected tip of the attacker and V is the virtual block of the honest node. Let D satisfy that $D \leq^\star C$ and $D <^\star V$ (such an element must exist, e.g. the genesis element). Then

$$\begin{aligned} adv(t) &= |BlueSet(C)| - |BlueSet(V)| \\ &= |BlueSet(C)\setminus past(D)| - |BlueSet(V)\setminus past(D)| \\ &\leq \max_{C'}\left\{|BlueSet(C)\setminus past(C')| - |BlueSet(V)\setminus past(C')|\right\} \end{aligned}$$

The analysis of the last term is similar to the analysis of $adv'(t)$ with two deviations: First, the growth rate of the honest network's blue score is not $(1-\alpha)$ rather $(1-\alpha)(1-\delta)$ as proven in Lemma 9. This has no qualitative effect on the analysis, it merely implies that the stationary distribution is governed by an exponent with base $\frac{\alpha}{(1-\alpha)(1-\delta)}$ rather than $\frac{\alpha}{(1-\alpha)}$. Secondly, the analysis therein assumes that no freeloading occurs. Fortunately, the Freeloader Bound guarantees that if the selected tip at time t was freeloading, $adv(t)$ is bounded by a constant $3k$. Therefore, by assuming the attacker always manages to saturate this constant, so that their advantage never goes below $3k$, we can shift the process $adv'(t)$ by $3k$ and analyze it as a block race, utilizing the result from [14]. □

A.5 The Markovian Process

In this section we prove the existence of the Markovian process described in Lemma 8. In order to do so, it is illuminating to first consider how one might prove the liveness property of GHOSTDAG in the non-adversarial setting. In this setting, it is easiest to demonstrate liveness by the emergence of so called *hourglass blocks*.

Definition 13. *A block B discovered in time t is called an* hourglass block *if it is the only block discovered between $t-2D$ and $t+2D$.*

Lemma 14. *In the non-adversarial setting where all miners are honest, if an honest hourglass block was created at time t, then the ordering of blocks created before time t will remain unchanged after t.*

Proof. Let B be the hourglass block created at time t. If B' was created before t, then by hypothesis it was created before $t-2D$, so by the definition of D it follows that $B' \in past(B)$ (as all miners are assumed to be honest). Similarly, if B'' was created after t it was created after $t+2D$ and is therefore in $future(B)$. It follows that B's anticone is empty, or equivalently that $B <^\star B''$ for any B'' created after B. Hence, the ordering over all blocks in created before t is precisely the ordering induced by B, which is deterministic and does not change. □

Corollary 15. *In the non-adversarial setting where all miners are honest, the probability that the ordering between two blocks created before time t will change at time $t+r$ decays exponentially with r.*

Proof. Let B, C be two blocks created before time t. At any point in time after t, the probability of the event E_t where the next block will be an hourglass block is some positive constant C depending only on the system parameters; moreoever, the events E_t and E_s are independent as long as $|s-t| > 2D$. As r grows, $|[t, t+r]|$ approaches $r\cdot\lambda$ exponentially fast. In particular, as r grows, with high probability (i.e., apart from a term in $O(e^{-F\cdot r})$) there's a set of blocks S of size at least $r\cdot f$ such that any two blocks in the set were created at least $2D$ seconds apart (F is some positive integer, $0 < f < 1$ is some positive constant). Since the events where blocks in S are hourglass blocks are mutually independent, we get that the probability that none of them is an hourglass block is at most $(1-c)^{f\cdot r}$. □

In the adversarial setting, hourglass events are not enough to secure the ordering. If the honest network posts an hourglass block B, the adversary could "undo" it by posting a block in the anticone of B. Honest blocks created after B might include this block in their blue set, thus making the ordering of blocks in the past of B uncertain again. Overcoming this requires a more specific event: we require that the honest network created a chain of several blocks while the adversary has produced none[14]. We call such an event a *burst*, and the blocks in the chain the *burst blocks*. We want to assure that the probability that the honest network is never convinced by a block whose selected chain does not include any of the burst blocks is bounded from below by some positive constant depending only on system parameters. In the following lemma, we prove that the amount of burst blocks required to achieve this depends only on system parameters and on $adv(t)$ where t is the time the burst started.

Lemma 16. *For any Z, there exists a constant time length s depending only on Z and system parameters, such that for any t there exists an event $D_{t,Z}$ which, if $adv(t) \leq Z$, implies that there exists a block B created before $t+s$ such that for the attacker to create a block C which satisfies that $B \not<^\star C$ and $adv(C) \geq -2k$ they must win a block race starting with advantage -1. Furthermore, the probability of $D_{t,Z}$ is bounded from below by a constant depending only on Z and system parameters.*

Proof. The event $D_{t,Z}$ is the event that no blocks were created during the first $2D$ seconds, after which the honest network created $Z'+3k+2$ blocks arranged in a chain, while the attacker created none, after which no blocks were created for $2D$ seconds, where $Z' = \max Z, k+1$. Assume furthermore that the entire event lasted at most $s = (2Z'+3k+2)/\alpha\lambda + 4D$ seconds (any $s > 4D$ would work, but this choice of s is reasonable as the probability for such an event lasting longer decreases exponentially with Z). Finally, let B the $3k+2$ from last block created in the event. We immediately note that the independence of $D_{t,Z}$ and $D_{t+s,Z}$ is true by design.

We call this event a *burst* and the blocks created during the event as *burst blocks*.

[14]Note that this is not a necessary condition for the network to converge, and hence the contansts which follow from the following analysis that are far from tight.

Yonatan Sompolinsky, Shai Wyborski, and Aviv Zohar

Let C_0 be an attacker block created after $t+s$ such that $B \not<^\star C_0$ and the selected parent of C_0 was created before $t+s$. Let $C = \max(SelectedChain(C_0) \cap \mathcal{A})$, then $adv(C) \leq Z$ and in particular $score(B) > score(C)$.

Let B' be a burst block created after B and assume that $C \in SelectedChain(B')$, this implies that $B <^\star B'$ since if C was discovered after B was created then any burst block would choose the previous burst block (which has score at least $score(B)$) over C as a selected parent.

This implies that all burst blocks created after B are in the anticone of C_0. Let B' be the latest burst block. It follows that $score(C_0) \leq score(B) + k$ so $score(B') \geq score(B) + 2k + 1 \geq score(C_0) + k + 1$. In particular, $adv(C_0) \leq -(2k-1)$.

Let E be a block created after $t+s$ which freeloads off $D \in future(c_0)$, but such that no block in $SelectedChain(E)$ freeloaded off a block in $futureC_0$.

Let E' be an element of $past(D) \cap \mathcal{A}$ of maximal score (this set is not empty as it contains the genesis block). If $score(E) - score(E') > k$ then there are more than k blocks in $BlueSet(E) \setminus past(D)$, which contradicts that $D \in BlueSet(C)$. Hence $score(E) - score(E') < k$. Note that $score(E') > score(E) - k > score(D) - k$. So

$$\begin{aligned} adv(E') &\geq score\left(E'\right) - (score(D) + k) \\ &\geq score\left(E'\right) - score(E) - k \\ &\geq -2k \end{aligned}$$

Since $adv(C) \leq -(2k-1)$ and no block below E freeloaded, this implies that the attacker has won a block race with starting with advantage -1.

□

We now prove Proposition 8, thereby concluding the proof of Theorem 7.

Proposition 8 *Assume that B convinced the network at time t, then $adv_B(t)$ is bound from above by a Markovian process on $\mathbb{Z}$ which goes to $-\infty$ exponentially fast.*

Proof. Let B_t be the event that both $D_{t,Z}$ and $adv(t) \leq Z$ hold. Lemma 10 implies that the probability of $adv(t) \leq Z$ is bound by a constant depending only on system parameters, while Lemma 16 states that the probability of $D_{t,Z}$ depends only on Z. It is evident from the construction of $D_{t,Z}$ that it depends only on blocks created after t while $adv(t) \leq Z$ depends only on block created after t, hence these events are independent, so the probability of their intersection is also bound by a constant.

In order to construct M we construct another process M' on $\mathbb{Z} \times \{0,1\}$ where the $\{0,1\}$ part keeps track on whether the attacker is in the condition described in 16. This means that as $adv_b < -k$, the attacker is in a block race. M is then defined as the projection of M' on $\mathbb{Z}$.

From Lemma 9 it follows that the honest chain grows at a rate of $(1-\alpha)(1-\delta)$ while the attacker chain grows at rate α. Let $\alpha' = \frac{\alpha}{1-\delta}$. This renormalizes things such that if the attacker does not freeload the advantage is modeled as a walk on $\mathbb{Z}$ which transitions right with probability α'.

We define transition probabilities of the state (n,i):

- If $n > Z$ then the process transitions to $(n+1,i)$ with probability α' or to $(n-1,i)$ with probability $1-\alpha'$
- If $-2k \leq n < 3k$ then the attacker may freeload, we assume that it always manages to do so and always gains the maximal advantage of $3k$, so (n,i) transitions to $(3k,1)$ with probability 1
- If $3k \leq n < Z$ then let p the probability of B_t, the process transitions to $(n+1,i)$ with probability $\alpha' p$, to $(n-1,i)$ with probability $(1-\alpha')p$, or to $(-k-1,0)$ with probability p
- If $n < -2k$ and $i = 1$ we also assume that the attacker manages to steal a block to gain the highest possible advantage, so the process always transitions to $(3k,1)$
- Finally, if $n < -2k$ and $i = 0$ then the attacker is not able to freeload as long as n stays below $-2k$, so the process transitions to $(n+1,0)$ with probability α' or to $(n-1,0)$ with probability $1-\alpha'$.

Note that the state $(-(2k+1),0)$ transitions to $(-2k,0)$ with probability α' from which it transitions to $(3k,1)$ with probability 1.

It is true by design that this process bounds the attacker advantage from above in the case where they insists not to include B in their selected chain (since all deviations from any actual process which is induced by any attacker strategy are positive), and that M goes to $-\infty$ exponentially fast.

□