PHANTOM:

A Scalable BlockDAG protocol

Yonatan Sompolinsky and Aviv Zohar

School of Engineering and Computer Science,
The Hebrew University of Jerusalem, Israel
{yoni_sompo,avivz}@cs.huji.ac.il

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 protocol for transaction confirmation that is secure under any throughput that the network can support. PHANTOM thus does not suffer from the security-scalability tradeoff which Satoshi's protocol suffers from. PHANTOM utilizes a Directed Acyclic Graph of blocks (a *blockDAG*), a generalization of Satoshi's chain which better supports faster block generation and larger blocks that take longer to propagate. PHANTOM uses a greedy algorithm on the blockDAG to distinguish between blocks mined properly by honest nodes and those that created by non-cooperating nodes who chose to deviate from the mining protocol. Using this distinction, PHANTOM provides a robust full order on the blockDAG in a way that is eventually agreed upon by all honest nodes.

1. Introduction

A. The Bitcoin protocol

The Bitcoin protocol instructs miners how to create blocks of transactions. The body of a block contains new transactions published by users, a proof-of-work puzzle, and a pointer to one previous block. The latter rule implies that blocks naturally form in a tree structure. When creating his next block, the miner is instructed to reference the tip of the longest chain within the tree and ignore the rest of blocks (aka *orphans*).

Miners share and propagate a block immediately upon receiving or creating it, and reference the latest block in the chain they observe. The security of Bitcoin relies on honest nodes being

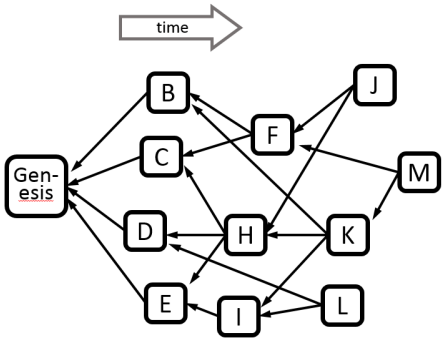

Fig. 1: An example of a block DAG G. Each block references all blocks to which its miner was aware at the time of its creation. The DAG terminology, applied to block H as an example, is as follows:
$past\,(H) = \{Genesis, C, D, E\}$ – blocks which H references directly or indirectly, and which were provably 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. **Deciding the order between H and blocks in $anticone\,(H)$ is the main challenge of a DAG protocol**.
$tips(G) = \{J, L, M\}$ – leaf-blocks, namely, blocks with in-degree 0; these will be referenced in the header of the next block

sufficiently connected so that when one miner extends the chain with a new block, it propagates in time to all honest nodes before the next one is created.

In order to guarantee this property, the creation of blocks is regulated by the protocol to occur once every 10 minutes. As a result, Bitcoin suffers from a highly restrictive throughput in the order of 3-7 transactions per second (tps).

B. The PHANTOM protocol

In this work we present PHANTOM, a protocol that enjoys a very large transaction throughput compared to Bitcoin. PHANTOM structures blocks in a Directed Acyclic Graph, a *blockDAG*. Rather than extending a single chain, miners in PHANTOM are instructed to reference all blocks in the graph (that were not previously referenced, i.e., leaf-blocks). An example of a blockDAG, and the basic terminology, is provided in Figure 1 above.

The core challenge of a DAG protocol is how to order transactions embedded in it, so that in case of two (or more) conflicting transactions, we can accept the one that arrived first (according to the prescribed order) and reject the other(s).

To that end, PHANTOM relies on the interconnectivity of honest nodes (similarly to Bitcoin's assumption in slow rates). Since cooperating PHANTOM miners propagate their blocks as soon as possible and reference all of their counterparts' blocks, we should expect to see in the DAG a well-connected cluster of blocks. In contrast, blocks that were mined by non-cooperating nodes will appear as outliers and will easily be discerned. Indeed, deviation from PHANTOM's mining protocol comes in the form of (i) withholding a new block for a while, or\and (ii) creating a new block that does not reference other blocks available at the time, both cases in which the new block can be recognized and penalized.

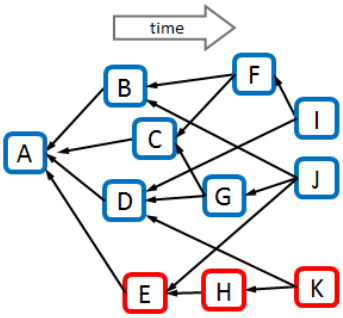

Fig. 2: An example of the largest 3-cluster of blocks within a given DAG: A, B, C, D, F, G, I, J (coloured blue). It is easy to verify that each of these blue blocks has at most 3 blue blocks in its anticone, and (a bit less easy) that this is the largest set with this property. Setting PHANTOM's inter-connectivity 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.

Following this intuition, together with the assumption that honest nodes hold a majority of the hashrate, we argue that the *largest* set of blocks with good inter-connectivity was mined by honest nodes, with high probability. Accordingly, given a blockDAG, we would want to solve the following optimization problem:

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

Here, $anticone\,(B)$ 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). The parameter k is related to an assumption that PHANTOM makes regarding the network's propagation delay; this is explained in detail in Section 4, following the formal framework specified in Section 3.

In practice, the Maximum k-cluster SubDAG is NP hard (see problem [GT26] in [2]). PHANTOM works therefore with a variant of this problem, using a greedy algorithm approach. The algorithm will be given in Sections 2, and some interesting variants in Section 6.

Once the set of honest and dishonest blocks are properly recognized by the protocol, we order the DAG in a way that favours the former set and penalizes the latter. Interestingly, the specific way in which we order honest blocks is of no importance to the security of the protocol—any arbitrary rule which respects the topology will enjoy the robustness properties of PHANTOM, as we prove formally in Section 5.

However, the ordering rule does affect confirmation times. An arbitrary topological ordering might take a long while before converging, especially if an active visible attack is taking place.

Thus, vanilla PHANTOM does not guarantee fast convergence time in all cases. In Section 7 we elaborate on this, and discuss techniques and modifications that allow fast confirmation times.

C. Related work

Many suggestions to improve Bitcoin's scalability have been proposed in recent years. These proposals fall into two categories, *on-chain scaling* and *off-chain scaling*. Roughly speaking, the former includes protocols where all valid transactions are those that appear – as in Bitcoin – inside blocks that are organized in some data structure (aka "the ledger").

On-chain scaling. The protocols in this category may differ e.g. in how fast blocks are created, how blocks are organized in the 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 [9], where a main chain of blocks is chosen according to a greedy algorithm and not through the longest chain rule; Inclusive [5], 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 [1], 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 [3]. The DAG in Iclusive 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 [8]. 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 full linear ordering, due to possible Condorcet cycles. PHANTOM solves this issue and provides a linear ordering over the blocks of the DAG. As such, PHANTOM can support consensus regarding any general computation, also known as *Smart Contracts*, which SPECTRE cannot. 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.[1] PHANTOM's linear ordering does not come without cost—confirmation times are mush slower than those in SPECTRE. In Section 7 we describe how the same system can simultaneously enjoy the best of both protocols.

Off-chain scaling. Another totally different approach keeps block creations infrequent and their sizes small (so that propagation delay remains negligible), yet this slow chain is not used for recording the entire economic activity. Instead, most of the transactions occur outside the chain, with better scalability, and the chain itself is used for the purpose of resolving conflicts or settling transactions. One example is Hybrid Consensus [6], improving over [4], which uses the chain to select a rotating committee of nodes which in turn run a classic consensus protocol to

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

confirm transactions in the corresponding epoch. Another well known proposed solution in the same category is the Lightning Network [7] (LN), where transactions are processed off-chain over over a network of micropayment channels, and the blockchain is used only for settlement of these channels.

Our work is orthogonal and complementary to these solutions, and can enhance their operation by orders-of-magnitude. For instance, when the DAG is used to serve channel-settlement transactions of LN, 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.

2. The PHANTOM protocol

In this section we describe the operation of the PHANTOM protocol. PHANTOM consists of the following three-step procedure:

1) Using the structure of the DAG, we recognize in it a cluster of well-connected blocks; with high probability, blocks that were mined honestly belong to this cluster and vice versa.
2) We extend the DAG's natural partial ordering to a full topological ordering 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 it. We iterate over all transactions in this order, and accept each one that is consistent (according to the underlying Consistency notion) with those approved so far.

The Consistency notion used in the last step depends on the specific application under consideration. For instance, with regards to the Payments application, a transaction is consistent with the history only if all of its inputs have been approved and no double spending transaction has been approved before. Our work is agnostic to the definition of the Consistency rule. The contribution of PHANTOM is its implementation of the first two steps described above, which we now turn to describe.

A. Intuition

How can we distinguish between honest blocks (i.e., blocks mined by cooperating nodes) and dishonest ones? Recall that the DAG mining protocol instructs a miner to acknowledge in its new block the entire DAG it observes locally, by referencing the "tips" of the DAG. Thus, if block B was mined at time t by an honest miner, then any block published before time $t - D$ was received by the miner and is therefore in B's past set (i.e., referenced by B directly or recursively via its predecessors; see illustration in Figure 1). Similarly, if B's miner is honest then it published B immediately, and so any honest block created after time $t + D$ belongs to B's future set.

As a result, the set of honest blocks in B's anticone – which we denote $anticone_{\boldsymbol{h}}(B)$ – is typically small, and consists only of blocks created in the interval $[t - D, t + D]$.[2] In other

[2] Note that, in contrast to $anticone_{\boldsymbol{h}}(B)$, an attacker can easily increase the size of $anticone(B)$, for any block B, by creating many blocks that do not reference B and that are kept secret so that B cannot reference them.

words, the probability that an honest block B will suffer a large honest anticone is small: $\Pr\left(|anticone_{\boldsymbol{h}}\left(B\right)| > k\right) \in \mathcal{O}\left(e^{-C \cdot k}\right)$, for some constant $C > 0$ (this stems from a bound on the Poisson distribution's tail). We rely on this property and set PHANTOM's parameter k such that the latter probability is smaller than δ, for some predefined $\delta > 0$; see discussion in Section 4.

Following this intuition, the set of honest blocks (save perhaps a fraction δ thereof) is guaranteed to form a k-cluster.

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

Note that the attacker can easily create k-clusters as well, e.g., by structuring his blocks in a single chain. Fortunately, we can leverage the fact that the group of honest miners possesses a majority of the computational power, and look at the *largest* k-cluster. We argue that the latter represents, in most likelihood, blocks that were mined properly by cooperating nodes. Refer to Figure 2 for an illustration of the largest 3-cluster in a given blockDAG. Identifying this set in general DAGs turns out to be computationally infeasible, and so in practice our protocol uses a greedy algorithm which is good enough for our purposes.

We term this selection of a k-cluster a *colouring of the DAG*, and use the colours blue and red as a convention for blocks inside and outside the chosen cluster, respectively.

B. Step #1: recognizing honest blocks

Algorithm 1 below selects a k-cluster in a greedy fashion. We denote by $BLUE_k\left(G\right)$ the set of blocks that it returns. The algorithm operates as follows:

1) Given a DAG G, the algorithm recursively computes on the past set of each tip in G.[3] This outputs a k-cluster for each tip.[4] (lines 4-5)
2) Then, it makes a greedy choice and picks the largest one among the outputted clusters. (lines 6-7)
3) Finally, it tries to extend this set and add to it any block whose anticone is small enough with respect to the set. (lines 8-10)

Intuitively, we first let the DAG inherit the colouring of its highest scoring tip, $B_{\max}$, where the score of a block is defined as the number of blue blocks in its past: $score\left(B\right) := |BLUE_k\left(past\left(B\right)\right)|$. Then, we proceed to colour blocks in $anticone\left(B_{\max}\right)$ in a way that preserves the k-cluster property. This inheritance implies that the greedy algorithm operates as a chain-selection rule—$B_{\max}$ is the chain tip, the highest scoring tip in $past\left(B_{\max}\right)$ is its predecessor in the chain, and so on. We denote this chain by $\mathcal{C}hn(G) = (genesis = \mathcal{C}hn_0(G), \mathcal{C}hn_1(G), \ldots, \mathcal{C}hn_h(G))$. The reasoning behind this procedure is very similar to that

[3] A tip is a leaf-block, that is, a block not referenced by other blocks. See Figure 1.

[4] Observe that, for any block B, the DAG $past\left(B\right)$ is fixed once and for all at B's creation, and in particular the set $BLUE_k\left(past\left(B\right)\right)$ cannot be later modified. Thus, in an actual implementation of Algorithm 1, the sets $BLUE_k\left(B\right)$ will have been computed already (by previous calls) and stored, and there will be no need to recompute them.

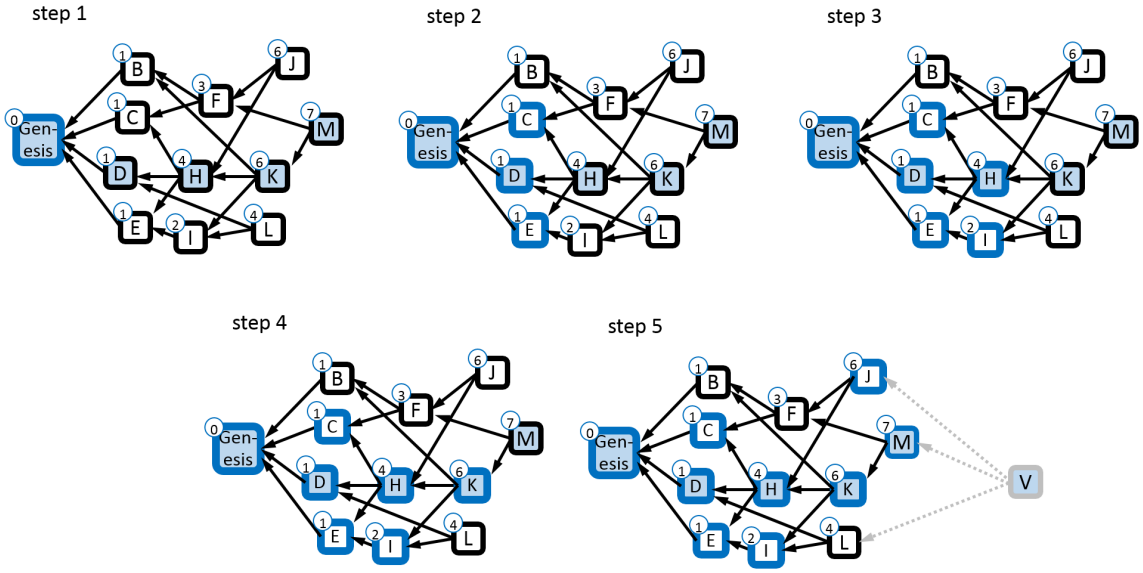

Fig. 3: An example of a blockDAG G and the operation of the greedy algorithm to construct its blue set $BLUE_k(G)$ set, under the parameter $k = 3$. The small circle near each block X represents its score, namely, the number of blue blocks in the DAG $past(X)$. The algorithm selects the chain greedily, 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 (it is 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 J to $BLUE_k(G)$, leaving L away due to its large blue anticone.

given in Section 1 in relation to the Maximum k-cluster SubDAG problem. They only differ in that, instead of searching for the maximal k-cluster, we are hoping to maximize it via the tip with maximal cluster and then adding blocks from its anticone. Thus, the reader should think of our algorithm (informally) as approximating the optimal solution to the Maximum k-cluster SubDAG problem.

We demonstrate the operation of this algorithm in Figure 3. Another example appears in Figure 4.

Note that the recursion halts because for any block $B \in G$: $|past(B)| < |G|$.

Let us specify the order in which blocks in $anticone(B_{\max})$ should be visited, in line 8 of the

algorithm. We suggest inserting all blocks in $anticone\left(B_{\max}\right)$ into a *lexicographical topological priority queue*, which we denote $topo\_queue$. The priority of a block is represented by the size of its past set;[5] in case of ties, the block with lowest hash ID is chosen.

Remark. *Our choice of ordering via $topo\_queue$ is not inherently significant, and many alternative topological orderings can provide similar robustness properties. That said, it might be the case that other rules provide faster convergence and confirmation times. We revisit this issue in Section 7.*

To summarize the function that the blue set satisfies, we state the following:

Proposition 2. *Let $G = G_{\infty}^{pub}$ be the eventual DAG containing all blocks in history, and let B be an arbitrary block in G.*

- *If B was created by an honest miner, the probability that B will not belong to $BLUE_k\left(G\right)$ decreases exponentially with k.*
- *If B was created by a malicious miner, and was withheld for a time interval of length T, the probability that B will belong to $BLUE_k\left(G\right)$ decreases exponentially with T.*

The proof of this proposition follows from the proof of Claim 3 in Section 5.

C. Step #2: ordering blocks

Recall that the DAG is ordered partially via its topology. We now wish to extend this order to a full order (aka *a topological order*). Our objective is to define a rule that gives precedence to blocks in the blue set, that penalizes blocks outside this set by deferring their location in the order, and that preserves the topological relations of blocks.

We propose the following procedure: Traverse the blue set according to some topological order, and iteratively add blocks to the current last position in ord^k; when visiting block B, first check if there are blocks in $past\left(B\right)$ that haven't been added yet, add such blocks to the order (again, according to some topological order), and then add B to the order.

For example, a possible output of ord^3 on the blockDAG illustrated in Figure 2 is: $(A, D, C, G, B, F, I, E, J, H, K)$.

This procedure is formalized in Algorithm 2 below. The algorithm begins by initializing an empty priority queue and an empty ordered list. Throughout, the list will represent the order in which blocks were popped out from the queue. The algorithm favours blocks in the blue set by adding to the queue all of the blue children of the current block. When a block is pushed into the queue, all blocks in its past (that weren't already treated) are pushed as well.

In the algorithm, $topo\_queue$ is the same lexicographical topological priority queue defined in the preceding subsection. In addition, the queue should avoid duplicating elements, and so $topo\_queue.push(C)$ should do nothing in case C is already in $topo\_queue$.

The intuition here is simple. The algorithm is intended to guarantee that a block B can precede a blue block C only if B is blue, or B is referenced by a blue block. In this way, blocks that

[5]This guarantees that a block cannot be popped out while a block in its past is still in the queue, since $C \in future\left(B\right) \Longrightarrow |past\left(C\right)| > |past\left(B\right)|$.

Algorithm 1 Selection of a blue set

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

function CALC-BLUE(G, k)
  if $B == genesis$ **then**
    return $\{genesis\}$
  for $B \in tips(G)$ **do**
    $BLUE_k(B) \leftarrow$CALC-BLUE($past(B), k$)
  $B_{\max} \leftarrow \arg\max\{|BLUE_k(B)| : B \in tips(G)\}$ (and break ties arbitrarily)
  $BLUE_k(G) \leftarrow BLUE_k(B_{\max}) \cup \{B_{\max}\}$
  for $B \in anticone(B_{\max})$ **do** in some topological ordering
    if $|anticone(B) \cap BLUE_k(G)| \leq k$ **then**
      add B to $BLUE_k(G)$
  return $BLUE_k(G)$

Algorithm 2 Ordering of the DAG

Input: G – a block DAG, k – the propagation parameter
Output: $ord(G)$ – an ordered list containing all of G's blocks

function ORDER(G, k)
  initialize empty queue $topo\_queue$
  initialize empty ordered list L
  $BLUE_k(G) \leftarrow$CALC-BLUE(G, k)
  $topo\_queue.push(genesis)$
  while $topo\_queue \neq \emptyset$ **do**
    $B \leftarrow topo\_queue.pop()$
    $L.add(B)$ (B is added to the end of the list)
    for all $C \in childrenB \cap BLUE_k(G)$ **do**
      for all $D \in past(C) \cap anticone(B) \setminus L$ **do**
        $topo\_queue.push(D)$
      $topo\_queue.push(C)$
  $ord(G) \leftarrow L$
  return $ord(G)$

were withheld by an attacker will not precede blocks that were mined properly and published on time (which are represented roughly by the set of blue blocks).

D. Implications to transaction security

We now demonstrate how the above procedures of PHANTOM enable safe acceptance of transactions. Consider a transaction $tx \in B$, where B is a block in the blue set of G. In order to render tx invalid, a conflicting transaction $\bar{tx}$ must precede it in the order, and must therefore be embedded in a block $C \in anticone\,(B)$ that precedes B.[6] The ordering procedure implies that, for C to precede B, it must either be a blue block or in the past set of a blue block. In both cases, C could not have been withheld for too long, by the second guarantee of Proposition 2. Thus, the recipient of tx can wait for the blue set around B to become sufficiently robust to reorgs, and then approve tx. In Section 5 we prove that robustness is indeed obtained, after some waiting time.

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 Bitcoin'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.

A. Network

We follow the model specified in [8]. 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 whatsoever 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 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.

[6] Or, $\bar{tx}$ can appear in B before tx, but this is not an interesting scenario.

B. 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.[7] 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). The attackers' computational power is less than 50%. Thus, $\sum_{v\in\mathcal{N}} \alpha_v = 1$, and $\sum_{v\in malicious} \alpha_v =: \alpha < 0.5$.

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 $G = (\mathcal{C}, E)$. Here, $\mathcal{C}$ represents blocks and E represents the hash references. We will frequently write $B \in G$ instead of $B \in \mathcal{C}$.

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.[8] 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 write $future\,(B)$ only when the context is clear or unimportant.

The set $anticone\,(B)$ represents all blocks not in B's future or past (excluding B as well). These are blocks whose ordering with respect to B is not defined via 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 illustrates this terminology.

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. $G_t^{oracle} := \cup_{v\in\mathcal{N}} G_t^v$ denotes the block DAG of a hypothetical oracle node, and $G_t^{pub} := \cup_{v\in honest} G_t^v$ denotes the block DAG containing all blocks that are visible to some honest node(s).

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:

[7] 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 $\mathcal{C}hn(G)$ grows by 2016 blocks.

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

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 $past\,(B) = G_t^v$.*[9]

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.

C. DAG client protocol

The DAG as described so far possibly embeds conflicting transactions. These are resolved on the client level. A *client* can be defined formally as a node in $\mathcal{N}$ which has no mining power. Intuitively, it is any user of the system who is interested in reading and interpreting the current state of the ledger.

In this work, a *transaction* tx is an arbitrary message that is embedded in a block. An underlying *Consistency* rule takes as input a set T of transactions and returns $valid$ or $invalid$. Our work is agnostic to the definition and operation of this rule, or to the characterization of the transaction space. Instead, we focus on the following task: devising a protocol through which all nodes agree on the order of all transactions in the system. Once such an order is agreed, one can iterate over all transactions, in the prescribed order, and approve each transaction that is consistent – according to the underlying rule – with those approved so far. *Such an ordering rule constitutes the client protocol*, and is run by each client locally without any need to communicate additional messages with other clients.

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 $C \notin G$.

D. 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 3. *Fix a rule ord. Let $B \in G = G_{t_0}^{pub}$. The function $Risk$ is defined by the probability that a block that did not precede B in time $t_1 \geq t_0$ will later come to precede it: $Risk\,(B, t_1) :=$ $\Pr\left(\exists s > t_1, \exists C \in G_s^{pub} : B \prec_{G_{t_1}^{pub}} C \wedge C \prec_{G_s^{pub}} B\right)$.*

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

[9]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.

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* converging *if* $\forall t_0 > 0$ *and* $B \in G^{pub}_{t_0}$*:* $\lim_{t_1 \to \infty} Risk\,(B, t_1) = 0$*, even when a fraction* α *of the mining power is byzantine.*

Remark. *Property 1 essentially couples the Safety and Liveness properties required from consensus protocols. Indeed, once* $Risk\,(B, t_1) < \epsilon$*, 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 Bitcoin and other protocols). 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^v_t*, and not merely from the public ledger's hypothetical perspective* G^{pub}_t*; 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 hashing power that the attacker must acquire i order to disrupt the protocol's operation:

Definition 4. *The* security threshold *of an ordering rule* ord *is defined as the maximal* α *(attacker's relative computational power) for which Property 1 holds true.*

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).

E. Main result

Our goal in this paper is to describe formally the ordering procedure of PHANTOM and to prove that it is scalable in the above sense.

Theorem 5 (PHANTOM 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 PHANTOM, parameterized with* $k(D_{max}, \delta)$*, is at least* $1/2 \cdot (1 - \delta)$*.*

The parameterization of PHANTOM via $k(D_{max}, \delta)$ is defined in the subsequent section (see (1)). Theorem 5 encapsulates the main achievement of our work. We prove the theorem formally in Section 5. Contrast this result to a theorem regarding the Bitcoin protocol, which appears in several forms in previous work (e.g., [6], [9]):

Theorem 6 (Bitcoin does not scale)**.** *As* λ *increases, the security threshold of the Bitcoin protocol goes to 0.*

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% to 0, deteriorating at a rate that depends on the error gap $D - D_{max}$.

4. Scalability and network delays

A. 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. 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 we assume that the unknown D is upper bounded by some D_{max} which is known to the protocol. The protocol does not explicitly encode D_{max}, rather, it is parameterized with k which depends on it, as will be described in the next subsection.

The use of an *a priori* known bound D_{max} distinguishes PHANTOM's security model from that of SPECTRE [8]. 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 PHANTOM 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 PHANTOM, but not in SPECTRE, should be ascribed to this added assumption; see further discussion in Section 7.

B. The anticone size parameter k

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

$$k(D_{max},\delta) := \min\left\{\hat{k}\in\mathbb{N} : \left(1-e^{-2\cdot D_{max}\cdot\lambda}\right)^{-1}\cdot\left(\sum_{j=\hat{k}+1}^{\infty} e^{-2\cdot D_{max}\cdot\lambda}\cdot\frac{(2\cdot D_{max}\cdot\lambda)^j}{j!}\right)<\delta\right\} \tag{1}$$

The motivation here is 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, at most $k(D_{max},\delta)$ additional blocks were created in the time interval $[t-D_{max}, t+D_{max}]$, with probability of at least $1-\delta$.[10]

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.

[10] In more detail: The second multiplicand in the definition of k bounds the probability that more than k blocks were created in parallel to B in the time interval $[t-D_{max}, t+D_{max}]$. This term is divided by $1-e^{-2\cdot D_{max}\cdot\lambda}$, which is the probability that at least one block was created during this time, namely, B. Thus, conditioned on the appearance of B, at most k blocks were created during this time interval, with probability of $1-\delta$ at least.

C. Trade-offs

Theorem 5, and the parameterization of PHANTOM in (1), tie between 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 5 states the security threshold in terms of δ. Following (1) we notice that tightening the security threshold – by choosing a lower δ – requires increasing k. A large k leads to slow confirmation times, as will be discussed shortly.[11]

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 remain on the safe side.[12] 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 PHANTOM'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, PHANTOM supports a very large throughput, and does not suffer from a security-scalability tradeoff.

That said, in PHANTOM 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 slow down the waiting time for transaction settlement. Thus, D_{max} should be set to a reasonable level. In Section 7 we discuss how this tradeoff can be restricted to visible conflicts only, and how applications such as *Payments* can enjoy very fast confirmation times nonetheless.

5. Proof

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

[11]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.

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

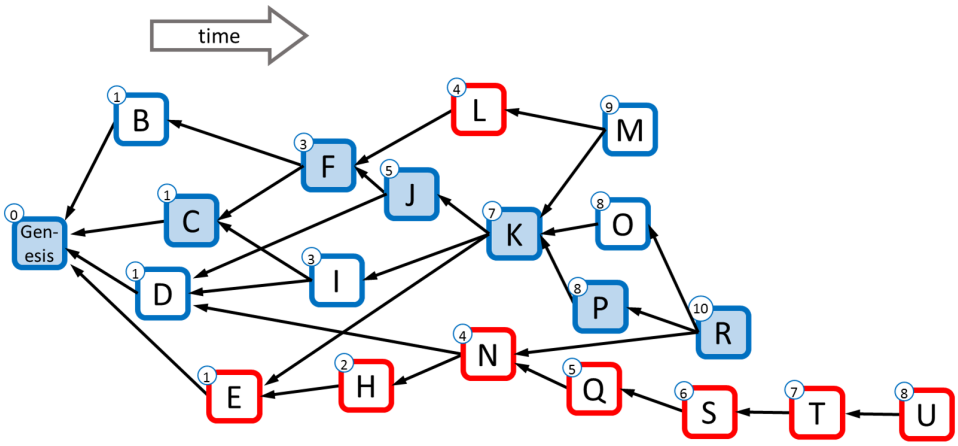

Fig. 4: An example of a DAG with an Hourglass block K. Here, the delay parameter is $k = 3$. As in Figure 3, the small circle near each block represents its score. The colouring of the DAG was done according to Algorithm 1. The greedily selected chain of blocks of highest score is marked with light-blue filling (note that this chain is not the longest one). Block K has the property that all blue blocks are either in K's past or in its future (in addition to K itself). It is thus an Hourglass block.

Theorem 5 (PHANTOM 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 PHANTOM, parameterized with $k(D_{max}, \delta)$, is at least $1/2 \cdot (1 - \delta)$.*

Our proof relies on the following lemma, which states that if some block B has the property that its anticone contains no blue blocks, then all blue blocks in its past precede all blocks outside its past. We call this the Hourglass property:

Lemma 7. *If for some $\widehat{B} \in G$, $BLUE_k(G) \cap anticone\left(\widehat{B}\right) = \emptyset$, then $\forall B \in past\left(\widehat{B}\right) \cap BLUE_k(G)$ and $\forall C \notin past\left(\widehat{B}\right)$, $B \prec_G C$. We then write $\widehat{B} \in Hourglass(G)$.*

In Figure 4 we provide an example of an Hourglass block. The proof of this lemma is straightforward from the operation of Algorithm 2:

Proof of Lemma 7. First note that if $BLUE_k(G) \cap anticone\left(\widehat{B}\right) = \emptyset$ then $\widehat{B} \in BLUE_k(G)$. Indeed, Algorithm 1 defines a chain, $\mathcal{C}hn(G)$ (see Subsection 2.2). This chain necessarily intersects some block in $anticone\left(\widehat{B}\right) \cup \left\{\widehat{B}\right\}$. And the intersection block must become blue itself, by lines (6-7). Thus, $BLUE_k(G) \cap anticone\left(\widehat{B}\right) = \emptyset$ implies that $\widehat{B} \in \mathcal{C}hn(G)$ and in particular $\widehat{B} \in BLUE_k(G)$.

Now, Algorithm 2 pushes a block into the queue only after it has pushed already all blocks in its past (lines 11 and 12). Therefore, $topo\_queue$ pops out blocks according to a topological order. Now, there are no blue blocks in $anticone\left(\widehat{B}\right)$, hence if C is a blue block then it must

belong to $future\left(\widehat{B}\right) \cup \left\{\widehat{B}\right\}$, in which case it must have been pushed to the queue after $\widehat{B}$ was popped (unless $C = \widehat{B}$). Thus, if C is blue, any block in $past\left(\widehat{B}\right)$ was popped out before C, and added to the ordered list L before it (line 8). In particular, $B \prec_G C$. On the other hand, if C is not blue, it was necessarily pushed into the queue only by belonging to the past set of some blue block B' (lines 9-11). B' was popped out after $\widehat{B}$ (unless $B = \widehat{B}$), from the same argument as in the previous case. Thus, C was pushed into the queue after $\widehat{B}$ was popped out, which in turn must have occurred after B was popped out. In particular, in this case as well, $B \prec_G C$. □

Lemma 8. *If $\widehat{B}$ is an Hourglass block in a DAG G, then G inherits the order from $\widehat{B}$: $ord(G) \cap past\left(\widehat{B}\right) = ord(past\left(\widehat{B}\right))$.*

Proof. In the proof of the previous lemma we have shown that $\widehat{B} \in \mathcal{C}hn(G)$. By the recursive operation of Algorithm 1 (lines 6-7), this implies that G inherits the colouring of $\widehat{B}$ on its past: $BLUE_k(G) \cap past\left(\widehat{B}\right) = BLUE_k\left(past\left(\widehat{B}\right)\right)$.

Now, no block in $anticone\left(\widehat{B}\right)$ is pushed into $topo\_queue$ before $\widehat{B}$ is popped out, because blocks get pushed in only if they are blue (and no such block exists in $anticone\left(\widehat{B}\right)$) or if a blue block in their future was pushed in—and this too only happens after $\widehat{B}$ is popped out. Since the queue respects the topology, this means that all blocks in $past\left(\widehat{B}\right)$ (which must be visited before $\widehat{B}$) are popped out before any block outside $past\left(\widehat{B}\right)$.

The fact that blocks in $anticone\left(\widehat{B}\right)$ are not pushed into the queue before all of $past\left(\widehat{B}\right)$ is popped out, implies further that the order in which Algorithm 2 pushes blocks in $past\left(\widehat{B}\right)$ into and out of $topo\_queue$ depends only on $past\left(\widehat{B}\right)$, and not on the DAG G. Hence, $ord(G) \cap past\left(\widehat{B}\right) = ord(past\left(\widehat{B}\right))$. □

The proof of Theorem 5 uses the occurrence of Hourglass blocks to secure all blocks in their past set.

Proof of Theorem 5. Let $\delta > 0$ and assume that $\alpha < 1/2 \cdot (1-\delta)$. We need to prove that $\forall t_0 > 0$ and $B \in G^{pub}_{t_0}$: $\lim_{t_1 \to \infty} Risk(B, t_1) = 0$, where $Risk(B, t_1) := \Pr\left(\exists s > t_1, \exists C \in G^{pub}_s : B \prec_{G^{pub}_{t_1}} C \wedge C \prec_{G^{pub}_s} B\right)$, and assuming a fraction of at most $1/2-\delta$ can deviate arbitrarily from the mining protocol.

Fix t_0 and B, and let $\tau(t_0)$ be the first time after t_0 where an honest block $\widehat{B}$ was published such that $\widehat{B}$ is an Hourglass block and will forever remain so:

$$\tau(t_0) := \min\left\{u \geq t_0 : \exists \widehat{B} \in G^{pub}_u \text{ such that } \widehat{B} \in \cap_{r \geq s} Hourglass\left(G^{pub}_r\right)\right\} \tag{2}$$

$\tau(t_0)$ is a random variable. By Lemma 9 it has finite expectation. In particular, $\lim_{t_1\to\infty} \Pr\left(\tau(t_0) > t_1\right) = 0$ (e.g., by Markov's Inequality).

Assume that such a $\tau(t_0)$ arrives, i.e., that a block $\widehat{B}$ is created and remains permanently an Hourglass block. Then, by Lemma 8, the order between all blocks in $past\left(\widehat{B}\right)$ never changes, and in particular the order between B and any other block C never changes. Thus, $= Risk\left(B, \tau(t_0)\right) = 0$. Therefore, $\lim_{t_1\to\infty} \Pr\left(\tau(t_0) > t_1\right) = 0$ implies $\lim_{t_1\to\infty} Risk\left(B, t_1\right) = 0$. □

Lemma 9. *Let $t_0 \geq 0$. The expected waiting time for $\tau(t_0)$ (defined in (2)) is finite, and moreover is upper bounded by a constant that does not depend on t_0.*

Proof. Let $\mathcal{E}(t_0)$ denote the event defined by the following conditions:

1) Some block $\widehat{B}$ was created at some time $u > t_0$ by an honest node (i.e., $\widehat{B} \in G_u^{pub}$) and apart from $\widehat{B}$ no other block was created in the time interval $[u - D, u + D]$.[13]
2) For some T_1, the k latest blocks in $BLUE_k\left(past\left(\widehat{B}\right)\right)$, denoted $LAST_k\left(past\left(\widehat{B}\right)\right)$, were created in the time interval $[u - T_1, u]$, and dishonest miners created no blocks in this time interval.
3) The score of $\widehat{B}$'s chain is forever higher than the score of any chain that does not pass through $\widehat{B}$: $\forall s \geq u, \forall C_1, C_2 \in tips\left(G_s^{pub}\right)$: $score\left(C_1\right) \geq score\left(C_2\right) \implies \widehat{B} \in \mathcal{C}hn(past\left(C_1\right))$.

Claim 2 and 4 together imply the required result.

The following claim states that the first two conditions in the above definition guarantee that as long as $\widehat{B}$ is blue no block in its anticone is blue:

Claim 1. *Assume that for some block $\widehat{B}$ the first two conditions in the definition of $\mathcal{E}(t_0)$ hold true. Then, as long as $\widehat{B}$ is blue, all blue blocks have $\widehat{B}$ in their past: $\forall s \geq u, \forall B \in anticone\left(\widehat{B}\right) : B \notin BLUE_k\left(G_s^{pub}\right) \vee \widehat{B} \notin BLUE_k\left(G_s^{pub}\right)$.*

Proof of Claim 1. Let $B \in anticone\left(\widehat{B}\right)$. If $\widehat{B} \notin BLUE_k\left(G_s^{pub}\right)$ we're done. Assume therefore that $\widehat{B} \in BLUE_k\left(G_s^{pub}\right)$. Any block that was published before time $u - D$ belongs to $past\left(\widehat{B}\right)$. Any block that was created after time $u + D$ by an honest node belongs to $future\left(\widehat{B}\right)$. As honest nodes did not create blocks in the interim, blocks in $anticone\left(\widehat{B}\right)$ can only belong to the attacker. Now, since the attacker did not create new blocks while blocks in $LAST_k\left(past\left(\widehat{B}\right)\right)$ were created, all attacker blocks that were created in the interval $[u - T_1, u]$ and that do not belong to $past\left(\widehat{B}\right)$ (hence belong to $anticone\left(\widehat{B}\right)$) have all blocks in $LAST_k\left(past\left(\widehat{B}\right)\right)$ in their anticone; formally: $\forall C \in anticone\left(\widehat{B}, G_u^{oracle}\right)$:

[13] We cannot assume, however, that no block was published during this interval, because the attacker might decide to publish during this interval blocks that he created earlier.

$LAST_k\left(past\left(\widehat{B}\right)\right) \subseteq anticone\left(C, G_u^{oracle}\right)$. Thus, if $\widehat{B} \in BLUE_k\left(G_s^{pub}\right)$, any block in $anticone\left(\widehat{B}\right)$ suffers an anticone that is larger than k (it contains $LAST_k\left(past\left(\widehat{B}\right)\right) \cup \left\{\widehat{B}\right\}$) and is therefore not in $BLUE_k\left(G_s^{pub}\right)$. In particular, $B \notin BLUE_k\left(G_s^{pub}\right)$.

Claim 2. *The waiting time for the event $\mathcal{E}(t_0)$ upper bounds $\tau(t_0)$.*

Proof of Claim 2. The previous claim implies that a block $\widehat{B}$ that satisfies the first two conditions is an Hourglass block in G_s^{pub}. The third condition implies that $\widehat{B}$ forever remains in the chain, and in particular forever remains blue. It is thus an Hourglass block in all DAGs G_s^{pub} ($s \geq u$).

Claim 3. *If the first two conditions in the definition of $\mathcal{E}(t_0)$ hold true then the third one holds true with a positive probability.*

Proof of Claim 3. Part I: We begin by assuming that the attacker did not publish any block in the time interval $[0, u - D_{max})$; formally, we assume that $\forall C \in past\left(\widehat{B}\right) : C \in honest$.

Let us compare the score of the honest chain to that of any chain that excludes $\widehat{B}$. This gap is captured by the following definition: For a time $r > 0$, define

$$X_r^1 := \max_{B:\widehat{B}\notin \mathcal{C}hn(B)} \{score\left(\mathcal{C}hn(B)\right)\} \tag{3}$$

$$X_r^2 := \max_{B:\widehat{B}\in \mathcal{C}hn(B)} \{score\left(\mathcal{C}hn(B)\right)\} \tag{4}$$

$$X_r := X_r^1 - X_r^2. \tag{5}$$

Let us focus first on the evolution of the process X_r between time 0 and time $u - T_1$. We refer to the lead X_{u-T_1} that the attacker obtained at the end of this stage as "the premining gap"; see [8]. Let B_1^r be the argmax of X_r^1, and let C_r^1 be the latest block in $G_r^{pub} \cap BLUE_k\left(past\left(B_1^r\right)\right)$, namely, the latest honest block which is blue in the attacker chain. Recall that for now we are assuming that all attacker blocks that were premined were kept secret until after time $u - D_{max}$. Observe that at most k blocks that were created by the attacker before $time\left(C_r^1\right)$ can be in $BLUE_k\left(past\left(B_1^r\right)\right)$ and can contribute to the score of the attacker's chain. Thus, between $time\left(C_r^1\right)$ and time $u - T_1$ – i.e., during the premining phase – the score of the attacker chain grows only via the contribution of attacker blocks, and therefore at a rate of $\alpha \cdot \lambda$ at most.[14]

Part II: Let us now consider the growth rate of the honest chain's score. Let $B_{\max}^t$ be the tip of the public chain $\mathcal{C}hn\left(G_t^{pub}\right)$. The score of this chain is by definition the score of this block.

[14]Notice that our analysis does *not* assume that the attacker creates its blocks in a single chain. We only claim that the attacker's highest scoring chain grows at a rate of $\alpha \cdot \lambda$ at most, because every attacker block can increase the attacker's highest scoring chain by 1 at most. Creating a single chain is indeed the optimal attack on the attacker side.

Now, by the choice of $k(D_{max}, \delta)$ (defined in 1), the probability of an arbitrary honest block B having too large of an honest anticone is small:[15]

$$\Pr_{B\sim\text{arbitrary honest block in } G_t^{pub}}\left(\left|\overline{anticone_h}\left(B, G_t^{pub}\right)\right| > k(D_{max}, \delta)\right) < \delta. \tag{6}$$

This is because block creation follows a Poisson process, and because honest blocks that were created D_{max} seconds before (after) B belong to its past (future), hence are not in its anticone. Since any block that is blue in the honest chain contributes to its score, at any time interval, the score of the public chain grows at a rate of $(1-\delta)\cdot(1-\alpha)\cdot\lambda$ at least. And at most k honest blocks created in the premining stage contributed to the score of the attack chain.

Part III: From Part I and Part II we conclude that the random process $X_r - k$ is upper bounded by the premining race analyzed in [8]. Therein it was shown that the probability distribution over $X_{u-T_1} - k$ (the process's state at the end of the premining stage) is dominated by the stationary probability distribution π of a reflecting random walk over the non-negative integers with a bias of $\frac{(1-\delta)\cdot(1-\alpha)}{1-\delta\cdot(1-\alpha)}$ towards negative infinity. The stationary distribution exists because $\alpha \leq 1/2\cdot(1-\delta) < (1-\alpha)\cdot(1-\delta)$.[16]

In particular, there is a positive probability that at time $u-T_1$ the premining gap, X_{u-T_1}, was less than k, so that $X_{u-T_1} - k < 0$.[17] Between times $u-T_1$ and u the honest network contributed additional $k+1$ to the score of the public chain, namely, blocks $LAST_k\left(past\left(\hat{B}\right)\right) \cup \left\{\widehat{B}\right\}$. During the same time the score of any secret chain(s) did not increase at all, per the second condition in the definition of $\mathcal{E}(t_0)$. Thus, $X_u = X_{u-T_1} - (k+1)$. And by the first assumption, $X_{u+D_{max}} = X_u$. All in all, with some positive probability, $X_{u+D_{max}} < 0$.

Part IV: Let us turn to look at the evolution of $(X_r)_{r\geq u+D_{max}}$ at the second stage. Assume that $X_{u+D_{max}} < 0$.

In Claim 1 we saw that for any r such that $\widehat{B} \in BLUE_k\left(G_r^{pub}\right)$, all blocks in $\widehat{B}$'s anticone are red in G_r^{pub}. This implies that, as long as $\widehat{B}$ is blue in the public DAG, only attacker blocks contribute to the score of the attacker's chain: $\widehat{B} \in BLUE_k\left(G_r^{pub}\right) \Longrightarrow \forall B \in anticone\left(\widehat{B}\right)$: $BLUE_k\left(past\left(B\right)\right) \setminus future\left(\widehat{B}\right) = \emptyset$ (indeed, note that all honest blocks created after time $u + D_{max}$ belong to $future\left(\widehat{B}\right)$). Consequently, the attacker's best chain grows at a rate of $\alpha\cdot\lambda$ at most, as this interval ($[u + D_{max}, \infty)$) as well, as long as $\widehat{B} \in BLUE_k\left(G_r^{pub}\right)$. We have already seen that at any time interval the honest chain's score grows at a rate of $(1-\delta)\cdot(1-\alpha)\cdot\lambda$ at least. Thus, starting at time $u+D_{max}$, the race between the honest chain's

[15]Below, $\overline{anticone_h}\left(B, G\right)$ denotes all blocks in $anticone\left(B, G\right)$ created by honest nodes.

[16]Note that we have multiplied $(1-\alpha)$ by $(1-\delta)$ to account for the rare events where there was a burst of block creation events and where consequently some honest blocks did not contribute to the score of the public chain. Note on the other hand that blocks created by honest nodes too do not contribute to the attacker chain's score, even if they are red in G_r^{pub}. This is because we argued that at most k blocks that were created by the attacker before $time\left(C_r^1\right)$ can be blue in $BLUE_k\left(past\left(B_1^r\right)\right)$, and this is regardless of C_r^1's status within G_r^{pub} (we merely used the fact that C_r^1 was created by an honest node, we didn't use the assumption that C_r^1 is blue in the honest chain).

[17]In fact, this probability is decreasing logarithmically as k increases.

score and the attacker chain's score can be modeled as a random walk over all integers with a bias of $\frac{(1-\delta)\cdot(1-\alpha)}{1-\delta\cdot(1-\alpha)}$ towards negative infinity.

Since the random walk begins at the negative location $X_{u+D_{max}}$, this supposedly implies that with a positive probability the walk will never return to the origin: $\Pr\left(\forall r \geq u + D_{max} : X_r < 0\right) > 0$. This in turn implies that $\widehat{B}$ will forever remain a blue block (and a chain block for that matter). However, to complete the analysis correctly, some careful attention is required:

Part V: First, we must account for the fact that not all honest nodes observe all honest blocks immediately, i.e., that G_r^v might be a proper subset of G_r^{pub}. This might give the attacker an advantage, as he can create blocks, reveal them immediately to honest nodes, and these blocks do not compete with honest blocks unseen yet by these nodes. This advantage can be accounted for by assuming that the race begins only after the attacker was given D_{max} additional seconds to create blocks while the honest network sat idle; see [8]. This fact does not change our general conclusion that $\Pr\left(\forall r \geq u + D_{max} : X_r < 0\right) > 0$, e.g., because there is a positive probability that the attacker did not create any block during these D_{max} additional seconds.

Secondly, observe that the honest chain grows according to a random process that is not necessarily Poisson. Fortunately, a result from [9] shows that nevertheless we can treat the block race as if the honest score grows according to a Poisson process, and that this assumption can only increase the value of X_r; it is thus a worst case analysis.

Part VI: Finally, we alleviate our assumption that the attacker published no block during the premining phase $[0, u - D_{max})$. Instead, consider any attacker block C that belongs to $past\left(\widehat{B}\right)$. If $C \in BLUE_k\left(past\left(BB\right)\right)$ then C contributed to the score of the honest chain which passes through $\widehat{B}$ and maybe to the attacker chain as well, so its existence does not change the above analysis. Similarly, if $C \notin BLUE_k\left(past\left(BB\right)\right)$, the fact that it was published has no consequence whatsoever on the score of the honest chain, and so we can ignore it and apply the same analysis as if it weren't published.

Claim 4. *The waiting time for the event $\mathcal{E}(t_0)$ is finite. Moreover, it is upper bounded by a constant that does not depend on t_0.*

Proof of Claim 4. Let B be an arbitrary honest block created in $time\left(\left(\right)B\right)$. The probability that no other block was created in the interval $[time\left(B\right) - D_{max}, time\left(B\right) + D_{max}]$ is given by $\left(1 - e^{-2\cdot D_{max}\cdot\lambda}\right)^{-1} \cdot \left(1 - \sum_{j=2}^{\infty} e^{-2\cdot D_{max}\cdot\lambda} \cdot \frac{\left(e^{-2\cdot D_{max}\cdot\lambda}\right)^j}{j!}\right)$. An arbitrary block B satisfies the first condition in the definition of $\mathcal{E}(t_0)$ with a positive probability.

Given an arbitrary block B satisfying the above condition, let T_1 be the creation time of the earliest block in $LAST_k\left(past\left(B\right)\right)$. The probability that the attacker did not create any block in the interval $[u - T_1, u - D_{max}]$ is given by $\left((1-\alpha)\cdot(1-\delta)\right)^k$. In particular, given the first condition, the second one is satisfied with a positive probability.

Importantly, for any two blocks B_1 and B_2 created after t_0 and that satisfy $|time\left(B_1\right) - time\left(B_2\right)| > 4 \cdot D_{max}$, the satisfaction of the first condition with respect to B_1 is independent from its satisfaction with respect to B_2. Consequently, the expected waiting time for the occurrence of a block $\widehat{B}$ which satisfies the first two conditions in the definition of $\mathcal{E}(t_0)$

is finite (see, for instance, Chapter 10.11 in [10]). Moreover, while the precise expected time $\mathbb{E}[Hourglass(t_0)]$ may theoretically depend on t_0, the above argument shows that $\mathbb{E}[Hourglass(t_0)] < const + 4 \cdot d$, where $const$ does not depend on t_0.

This completes the proof of Claim 4 and of Lemma 9. □

Theorem 5 guarantees that the probability of reorg with respect to a given block B diminishes: $Risk\,(B, t_1) \to 0$. However, it does not guarantee anything about the convergence rate, i.e., the waiting time for a t_1 that satisfies $Risk\,(B, t_1) < \epsilon$, for some $\epsilon > 0$.[18] Following the analysis in the proof of Claim 4, the waiting time for an Hourglass block can be upper bounded by a ***constant*** in the order of magnitude of $\mathcal{O}\left(e^{C \cdot D_{max} \cdot \lambda}\right)$, for some $C > 0$; and after such a block is created,the analysis implies that $Risk\,(B, t_1)$ converges to 0 at an exponential rate, due to the random walk dynamic.

However, the analysis used in the proof is not tight. It relies on rather infrequent events which guarantee convergence in a straightforward way, namely, on the occurrence of $Hourglass$ blocks. In practice the network is likely to converge much faster. Moreover, it can be shown that when there is no active attack, the convergence time is faster by orders-of-magnitude.

6. VARIANTS

In Section 2 we described the PHANTOM's greedy algorithm to mark blocks as blue or red (Algorithm 1). In fact, similar algorithms can provide similar guarantees. We describe several such variants below and explain the intuition behind them. All these variants can be thought of as greedy approximations to the Maximum k-cluster SubDAG problem described in Section 1.

A. Choosing the maximizing tip

In our original version of the colouring procedure, we chose the tip which has the highest score (Algorithm 1, line 6). Instead, the algorithm below chooses the tip for which the score of the (virtual block of the) current DAG would be highest:

B. Adding blocks to the greedily chosen blue set

Another variant over the previous algorithms is the procedure to mark more blocks as blue, after the best tip and its blue set were selected. Previously, we required that the candidate block admit an anticone of size at most k *in the current set*. Instead, we can require that its anticone be counted only inside the originally chosen blue set. Thus, yet another variant is to replace lines 8-9 with

Observe that the resulting blue set may contain blocks which have an anticone larger than k within the set.

Yet another viable alternative is to further relax the condition (in lines 8-9) by counting the anticone of the candidate B only against the resulting chain of blue blocks, $\mathcal{C}hn(past\,(B)) \cup \{B\}$, where B is the selected tip:

[18] ϵ represents the risk that the party confirming the transaction is willing to absorb.

Algorithm 3 Selection of a blue set

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

```
1: function CALC-BLUE(G, k)
2:     if B == genesis then
3:         return {genesis}
4:     for B ∈ tips(G) do
5:         BLUE_k(B) ←CALC-BLUE(past(B), k)
6:         S_B ← BLUE_k(B) ∪ {B}
7:         for C ∈ anticone(B) in some arbitrary order do
8:             if |anticone(C) ∩ S_B| ≤ k then
9:                 add C to S_B
10:    return arg max{|S_B| : B ∈ tips(G)} (and break ties arbitrarily)
```

```
8: if |anticone(C) ∩ (BLUE_k(B) ∪ {B})| ≤ k then
9:     add C to S_B
```

Compare this variant to Satoshi's longest-chain rule, which can be describe as follows: each block in a chain $\mathcal{C}hn$ increases its weight by 1, and we select the highest scoring chain. In contrast, in our last variant, each block whose gap from a chain $\mathcal{C}hn$ is at most k increases its weight by 1, and we select the highest scoring chain.

C. Iterative elimination of blocks

We now introduce another variant 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 all blue blocks admit a blue anticone of size k at most:

We conjecture that Algorithm 1 can be replaced with each of the greedy algorithms described in this section. To this end, suffice it to repeat the proof of Theorem 5 with respect to these variants. We suspect that the same proof technique, namely, the use of Hourglass, would prove useful.

7. CONFIRMATION TIMES

As discussed in Section 5, the convergence rate of $Risk(B,t)$ is slow, at least theoretically and under certain circumstances. Recall that the function $Risk(B,t)$ measures the probability

```
8: if |anticone(C) ∩ (Chn(past(B)) ∪ {B})| ≤ k then
9:     add C to S_B
```

Algorithm 4 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 $BLUE_k(G)$

that a certain block that did not precede B at time t will later come to precede it. Recall further that throughput this work we used arbitrary topological orderings (over blue blocks). In light if this, it would be interesting to seek for an ordering rule (over blue blocks) that would converge faster. We suspect this is not a trivial task, and leave its full investigation to future work.

The primary factor to the fact that PHANTOM cannot guarantee fast confirmation times is that membership in the blue set takes time to finalize. The waiting time for such finalization can be further increased if an attacker manages to balance the decision between $B \in BLUE_k(G)$ and $B \notin BLUE_k(G)$. Observe however that if a certain transaction $tx \in B$ admits no conflicts in $anticone(B)$, then tx can be accepted even before the decision regarding B is finalized.

A. Combining SPECTRE and PHANTOM

SPECTRE is a DAG based protocol that can support large transaction throughput (similarly to $PHANTOM$) and very fast confirmations. SPECTRE does *not* output a linear order over blocks. Rather, every block B admits a vote regarding the pairwise ordering of any two blocks C and D, and the output is the majority vote regarding each pair. In SPECTRE, cycles of the sort "B precedes C, C precedes D, D precedes B" may form.

Furthermore, if an active balancing attack is taking place, for *some* pairs of blocks (B, C) the SPECTRE relation might not become robust (in which case $Risk(B, t) \to 0$ is not guaranteed). Instead, a published block B is guaranteed to robustly precede any block C that was published later than B, unless C was published shortly after B. We call this property *Weak Liveness*. In the context of the Payments application, this fact can only harm a user that signed and published two conflicting payments at approximately the same time.[19]

Since $PHANTOM$ does guarantee (strong) Liveness and a liner ordering, it is interesting to inquire whether we can enjoy the best of both worlds. We provide below a partial answer to this question.

Consider the following procedure: Given a blockDAG G,

1) mark blocks as blue or red according to PHANTOM's colouring procedure

[19]In contrast, usually any user can engage with a smart contract and introduce conflicts inputs. Thus, Weak Liveness might potentially harm the usability of SPECTRE to the Smart Contracts application.

2) run SPECTRE on the subDAG $BLUE_k(G)$; this determines the pairwise ordering between any two *blue* blocks B and C
3) for any blue block B and red block C, if $C \in past(B)$ then determine that C precedes B, otherwise determine that B precedes C
4) decide the pairwise ordering of any two red blocks in some arbitrary way that respects the topology ($B \in past(C) \Rightarrow B$ precedes C)

Intuitively, we run SPECTRE on the set of blue blocks (as determined by PHANTOM), and complemented the pairwise ordering in a way that both penalizes red blocks and respects the topology. We argue that this protocol enjoys SPECTRE's fast confirmation times and at the same time inherits the (regular) Liveness property of PHANTOM. To see the latter, observe that a Hourglass block would have a similar effect in SPECTRE—all blocks in its future will vote according to its vote. In particular, the pairwise relation of all previous blocks becomes as robust as the Hourglass block.

Note that this incremental improvement over vanilla SPECTRE is only possible because we allowed the protocol to assume something on the network's topology, namely, that the communication delay diameter is upper bounded by D_{max}.

B. Summary

In summary, it is possible to achieve both fast confirmation times and Liveness by combining PHANTOM and SPECTRE. It is of yet unclear whether we can further achieve a linear ordering without compromising the fast confirmation times. Hopefully, we will provide answers to these questions in future work.

REFERENCES

[1] 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.

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

[3] Aggelos Kiayias and Giorgos Panagiotakos. On trees, chains and fast transactions in the blockchain. Cryptology ePrint Archive, Report 2016/545, 2016.

[4] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 279–296. USENIX Association, 2016.

[5] 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.

[6] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model, 2016.

[7] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. *Technical Report (draft)*, 2015.

[8] Yonatan Sompolinsky, Yoad Lewenberg, and Aviv Zohar. Spectre: A fast and scalable cryptocurrency protocol. *IACR Cryptology ePrint Archive*, 2016:1159, 2016.

[9] 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.

[10] David Williams. *Probability with martingales*. Cambridge university press, 1991.