PHANTOM, GHOSTDAG:

Two Scalable BlockDAG protocols

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 PoW 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 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 total order on the blockDAG in a way that is eventually agreed upon by all honest nodes.

1 INTRODUCTION

The security of the Bitcoin protocol relies on blocks propagating quickly to all miners in the network [1], [3], [6]. 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.

The main achievement of PHANTOM can be summarized as follows:

Theorem 1 (Informal)**.** *Given two transactions tx_1, tx_2 that were published and embedded in the blockDAG at some point in time, the probability that PHANTOM'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 upload soon a version with a formalization of this theorem and a proof. The security property given in the theorem allows PHANTOM to support a high block creation rate. This translates both to fast confirmation times and to a larger throughput of transactions.

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, are 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.

We now proceed to describe the PHANTOM protocol more formally.

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.

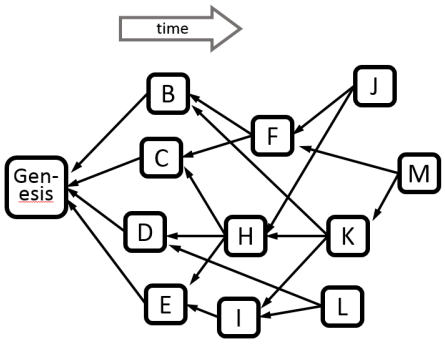

Fig. 1: An example of a block DAG G. Each block references all blocks that were known to its miner at the time it was created. The DAG terminology applies to H as follows:
$past\,(H) = \{Genesis, C, D, E\}$ – blocks which H references directly or indirectly, and which were provably created before H;
$future\,(H) = \{J, K, M\}$ – 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 The PHANTOM protocol

2.1 Preliminaries

The following terminology is used extensively throughout this paper. A DAG of blocks is denoted $G = (\mathcal{C}, E)$, where $\mathcal{C}$ represents blocks and E represents the hash references to previous blocks. We frequently write $B \in G$ instead of $B \in \mathcal{C}$. $past\,(B, G) \subset \mathcal{C}$ denotes the subset of blocks reachable from B, and similarly $future\,(B, G) \subset \mathcal{C}$ 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 $anticone\,(B, G)$.

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 2. *Given a DAG $G = (\mathcal{C}, E)$, a subset $S \subseteq \mathcal{C}$ 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 $G = (\mathcal{C}, E)$
Output: A subset $S^* \subseteq \mathcal{C}$ 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]

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 [2]), 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 cluster using a greedy algorithm. The algorithm constructs the Blue set of the DAG

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.

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 [6], called the GHOST protocol. We thank Ethan Heilman for suggesting this name.

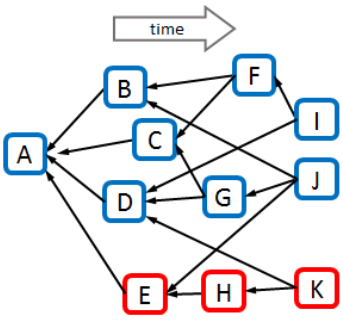

Fig. 2: An example of the largest 3-cluster of blocks within a given DAG: A, B, C, D, F, G, I, J (coloured blue). It is easy to verify that each of these blue blocks has at most 3 blue blocks in its anticone, and (a bit less easy) that this is the largest set with this property. Setting PHANTOM's inter-connectivity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

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 to the Blue set blocks outside $B_{\max}$'s past, provided that the k-cluster property is preserved.

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 $\mathcal{C}hn(G) = (genesis = \mathcal{C}hn_0(G), \mathcal{C}hn_1(G), \ldots, \mathcal{C}hn_h(G))$.

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: ord – an ordered list containing all blocks in G; $BLUE_k\,(G)$ – the Blue set of G

1: **function** ORDERDAG(G, k)
2: **if** $G == \{genesis\}$ **then**
3: **return** $[\{genesis\}, \{genesis\}]$
4: **for** $B \in tips(G)$ **do**
5: $[OrderedList_B, BlueSet_B] \leftarrow$OrderDAG$(past\,(B), k)$
6: $B_{\max} \leftarrow \arg\max\{|BlueSet_B| : B \in tips(G)\}$ (and 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** $[OrderedList_G, BlueSet_G]$

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

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.

longest-chain rule to a setup where block rate is high and propogation delays are not negligible.

In Section 5 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 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.

3.1 Network

We follow the model specified in [5]. 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.

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] 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

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

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.[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 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:

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*

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

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.

3.3 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 $C \notin G$.

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 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

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.

(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* converging *if* $\forall t_0 > 0$ *and* $B \in G^{pub}_{t_0}$*:* $\lim_{t_1 \to \infty} Risk\left(B, t_1\right) = 0$*, 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* $Risk\left(B, t_1\right) < \epsilon$*, 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).

3.5 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., [4], [6]):

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

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. 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 [5]. 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.

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:

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

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 $1 - \delta$.[9]

9. In more detail: The term written inside the definition of k bounds the probability that more than k blocks were created in parallel to B in the time interval $[t - D_{max}, t + D_{max}]$. Thus, with probability of at least $1 - \delta$, at most k blocks were created during this time interval (in addition to B).

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.

4.3 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.[10]

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.[11] 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.

10. 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.

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

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

This section is under construction. Will upload soon a version with correct proof.

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 below the two most interesting ones, and explain the intuition behind them. These variants can be thought of, *informally*, as greedy approximations to the Maximum k-cluster SubDAG problem described in Section 1.

6.1 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, Algorithm 2 below chooses the tip for which the score of the (virtual block of the) current DAG would be highest:

Algorithm 2 Ordering the DAG

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

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

6.2 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 we arrive at a k-cluster:

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

1: **function** CALC-BLUE(G, k)
2: $BLUE_k(G) \leftarrow V$
3: **while** $\exists B \in BLUE_k(G)$ with $|anticone(B) \cap BLUE_k(G)| > k$ **do**
4: $C \leftarrow \arg_B \max \{|anticone(B) \cap BLUE_k(G)|\}$ (with arbitrary tie-breaking)
5: remove C from $BLUE_k(G)$
6: **return** $BLUE_k(G)$

We conjecture that Algorithm 1 can be replaced with each of the greedy algorithms described in this section. We leave this conjecture for future work.

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 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.

7.1 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.[12]

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
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}.

7.2 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.

12. 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.

REFERENCES

[1] 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.

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

[3] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. *IACR Cryptology ePrint Archive*, 2016:454, 2016.

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

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

[6] 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.

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