The DAG KNIGHT Protocol: A Parameterless Generalization of Nakamoto Consensus

February 23, 2023

Michael Sutton
msutton@cs.huji.ac.il
Independent Researcher

Yonatan Sompolinsky
ysompolinsky@fas.harvard.edu
Harvard University

ABSTRACT

In 2008 Satoshi wrote the first permissionless consensus protocol, known as Nakamoto Consensus (NC), and implemented in Bitcoin. A large body of research was dedicated since to modify and extend NC, in various aspects: speed, throughput, energy consumption, computation model, and more [4]. One line of work focused on alleviating the security-speed tradeoff which NC suffers from by generalizing Satoshi's blockchain into a directed acyclic graph of blocks, a block DAG. Indeed, the block creation rate in Bitcoin must be suppressed in order to ensure that the block interval is much longer than the worst case latency in the network. In contrast, the block DAG paradigm allows for arbitrarily high block creation rate and block sizes, as long as the capacity of nodes and of the network backbone are not exceeded. Still, these protocols, as well as other permissionless protocols, assume an *a priori* bound on the worst case latency, and hardcode a corresponding parameter in the protocol. Confirmation times then depend on this worst case bound, even when the network is healthy and messages propagate very fast. In this work we set out to alleviate this constraint, and create the first permissionless protocol which contains no *a priori* in-protocol bound over latency. KNIGHT is thus responsive to network conditions, while tolerating a corruption of up to 50% of the computational power (hashrate) in the network. To circumvent an impossibility result by Pass and Shi [16], we require that the client specifies locally an upper bound over the maximum *adversarial* recent latency in the network. KNIGHT is an evolution of the PHANTOM paradigm [20], which in turn is a parameterized generalization of NC.

1 INTRODUCTION

The first permissionless consensus protocol, Nakamoto Consensus (NC), was created in 2008 by Bitcoin's originator Satoshi Nakamoto [13]. Permissionless is defined as an environment where the set of participants is not *a priori* known and fixed. Since its introduction, the research community offered many variants that improve upon NC in terms of speed, throughput, energy consumption, computation model, and more [4].

One line of work focused on alleviating the speed-security tradeoff, by generalizing Satoshi's blockchain into a directed acyclic graph of blocks – a block DAG [11, 19, 20]. Whereas in NC each block references a single predecessor, and a single chain within the resulting tree is extended, in DAG-based constructions blocks reference multiple predecessors. Blocks are thus created much more frequently than Bitcoin's 10 minutes interval, typically multiple blocks per one unit of network delay. This asynchronous operation mode opens up the possibility of conflicts across blocks created in parallel. The heart of the consensus protocol is its conflict resolution rule, which is written in the form of a DAG ordering algorithm—each node runs locally a procedure that takes as input the block DAG visible to it and returns a linear ordering over its blocks, and by implication over its transactions. This ordering ensures and recovers consistency: The first of any set of conflicting transactions is accepted, and the rest are ignored and skipped over. As any consensus protocol, this procedure must satisfy the property that all nodes eventually agree on the ordering.

KNIGHT is a parameterless DAG-based consensus—the protocol assumes no upper bound on the network's latency. In other words, the ordering procedure of KNIGHT does not take as input parameters representing the network's assumed latency. To the best of our knowledge, KNIGHT is the first permissionless parameterless consensus protocol that is secure against any attacker with less than 50% of the computational power in the network. These properties put KNIGHT at an inherently stronger spot than its counterparts: It is both faster and more secure, since it makes fewer assumptions and operates properly despite varying network conditions.

1.1 KNIGHT optimization framework

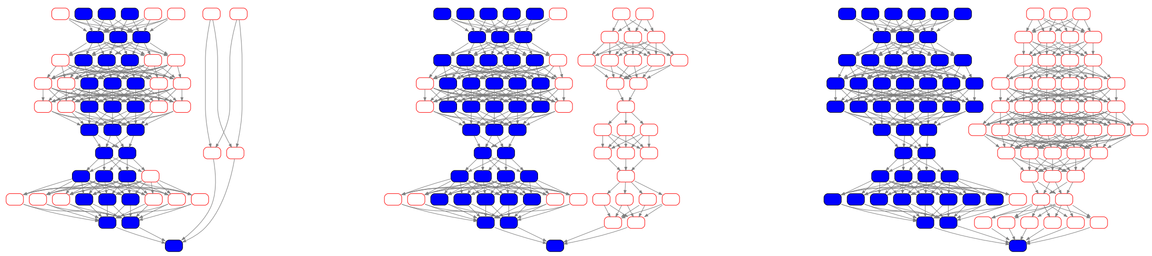
Conceptually, KNIGHT is an evolution of the PHANTOM optimization framework [20], which in turn is an evolution of NC. In NC, the longest chain of blocks within the tree is selected and extended. PHANTOM generalizes the longest chain rule: Rather than selecting the longest chain, it selects the largest sufficiently connected subset of blocks. The following definition from [20] captures "well-connectedness":

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

Here, the anticone of a block is the set of blocks whose order with respect to it is not dictated by the DAG topology; see Figure 2. Formally, PHANTOM solves the following optimization problem:

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

Similarly to other parameterized consensus protocols, the parameter k of PHANTOM represents an upper bound on the network's latency (technically, on the number of blocks per one unit of delay, with high probability). Observe that for $k = 0$, PHANTOM coincides with NC, as the longest chain is the largest 0-cluster. Indeed, when the block interval is large (e.g., Bitcoin's 10 minutes per block), the latency parameter k can be set to 0. In contrast, a system enjoying a



(a) $\alpha = 0.05, k = 2$ **(b)** $\alpha = 0.25, k = 4$ **(c)** $\alpha = 0.45, k = 7$

Figure 1: KNIGHT optimization illustrated. The figures show a fixed honest DAG on the left, alongside an attacker on the right. The attacker's byzantine fraction grows from $\alpha = 0.05$ in 1a to $\alpha = 0.45$ in 1c. The algorithm finds a k-cluster of minimum width ($k+1$) sufficient to cover at least 50% of the DAG, thereby outweighing the minority attacker.

high block creation rate would require setting k to be much larger. For instance, in Kaspa, a cryptocurrency based on PHANTOM, the block interval was set to 1 second, and k was hardcoded with a value of 18, reflecting an assumption of $D \leq 10$ seconds; see [2] for a live visualization of the live DAG of Kaspa.

In contrast, KNIGHT offers an alternative optimization framework, which does not pre-assume a latency bound:

KNIGHT Optimization: Minimal k Majority Cluster sub-DAG (*MkMC*)
Input: DAG $G = (C, E)$
Output: A subset $S^\star = MCS_k(G)$, s.t. k is minimal and $|S^\star| \geq \frac{|C|}{2}$.

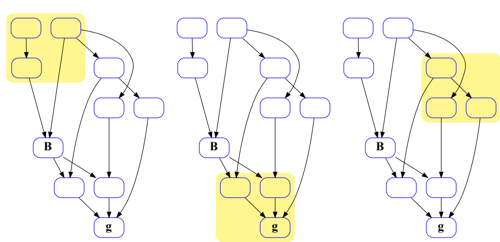


Figure 2: The topology of a block-DAG induces a partial ordering over blocks. The figure on the left marks blocks provably created after block B, which are called its *future* set. Similarly, the figure on the middle marks blocks provably created before B, its *past* set. The right-most figure marks blocks whose ordering with respect to B is ambiguous and must be dictated and agreed by the consensus protocol.

That is, rather than selecting the largest k-cluster for one predetermined value of k, we select the largest k-cluster for each value of k, and pick the minimal k whose maximizing cluster covers 50% of the DAG. We thus utilize the honest majority assumption to recognize a subset of blocks that are sufficient to counter an attack. In this way, we avoid the need to know k in advance, and allow the protocol to self-adjust to the real time latency. The actual KNIGHT protocol contains more components than the optimization problem *MkMC*, not merely for efficiency but also for security considerations – primarily, natural or malicious changes in the latency[1] – as will be described formally in Section 2. See Figure 1 for an illustration of KNIGHT's optimization dynamics.

1.2 Parameterlessness

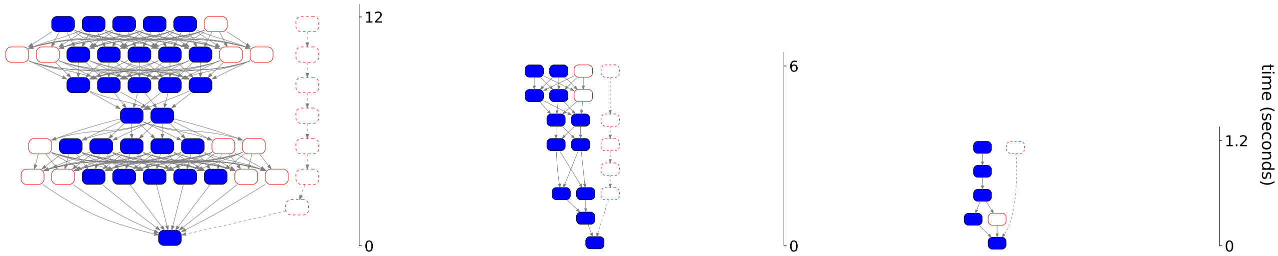
KNIGHT differs from previous work on proof-of-work-based consensus protocols which typically operate in the synchronous setup and assume an *a priori* upper bound over D, either explicitly or implicitly. For instance, Bitcoin's difficulty adjustment algorithm is targeting a block creation rate of $\lambda = 1/600$ blocks per second, which reflects an underlying assumption that $D \ll 600$ seconds. Similarly, when instantiating the PHANTOM protocol, one must pre-configure the protocol's k parameter which represents the expected number of blocks created in one unit of delay, reflecting an assumption that $D \ll \frac{k+1}{\lambda}$.

Parameterlessness has two implications. First, confirmation times in parameterized protocols are typically limited by their parameter—they are a function of the hardcoded parameter, regardless of the network's actual latency. Thus, even when the actual latency of blocks in Bitcoin is 1 or 2 seconds (as is the situation for most of the time, see [1]), the protocol's convergence times is in the order of tens of minutes. Similarly, Kaspa's convergence time remains in the order of tens of seconds even when its latency is way below 10 seconds.

As a parameterless protocol, KNIGHT avoids this shortcoming and allows the network to converge according to its actual conditions. Thus, when the network's *adversarial* latency is very low, the ordering of KNIGHT will converge immediately, allowing clients to confirm transactions within a few Internet RTTs (Round Trip Times); and when the network is slow and clogged, the ordering will take longer to converge and transactions longer to confirm. Crucially, we emphasize that this responsiveness is with respect to the worst-case adversarial latency; in Subsection 1.4 we distill this nuance.

Figures 3 and 4 demonstrate this effect. In the former, a DAG of various "widths" is presented, corresponding to different network latencies. When the network is speedy, miners are aware of almost

[1] When the delay is roughly constant, KNIGHT coincides with PHANTOM, and in particular when the delay is negligible, it coincides with NC.



(a) $D = 2, k = 4,$ **conf. time** $= 12$

(b) $D = 1, k = 1,$ **conf. time** $= 6$

(c) $D = 0.1, k = 0,$ **conf. time** $= 1.2$

Figure 3: Confirmation times of KNIGHT under various network conditions. As the latency decreases from 2 seconds per block message in (3a) to 0.1 seconds in (3c), k decreases from 4 to 0, and confirmation times improve from 12 to 1.2 seconds. Observe that the algorithm correctly recognizes a cluster of minimal width ($=k+1$) which still covers at least 50% of the corresponding DAGs ($k = 4, 1, 0$ in 3a, 3b, 3c, respectively) as required by KNIGHT's optimization framework. The depicted DAGs are a result of a simulation of a network mining $\lambda = 3.75$ blocks per second and an (invisible, secret) attacker with $\alpha = 0.2$; the required confidence parameter was set to $\epsilon = 0.05$.

all blocks created by their peers, blocks enjoy small anticones (or "gaps") of size 1 at most, and transactions can be confirmed quickly. On the other extreme, many blocks are created in parallel, blocks suffer from larger anticones, and transactions take longer to confirm. This scenario represents either a slow down in the network, or a system intentionally parameterized with a high block rate, e.g., $\lambda = 10$ blocks per second. Figure 4 further compares the effect of varying network conditions on parameterized protocols (e.g., NC, PHANTOM) and parameterless ones (e.g., KNIGHT). The confirmation times in the former protocols are constant, accounting to the hardcoded latency-dependent parameter; worse yet, when the network suffers an anomaly, and message delays violate the bound, transactions cannot be confirmed altogether. In contrast, the confirmation times of parameterless protocols correspond to the (bound of the client over the maximum) current latency in the

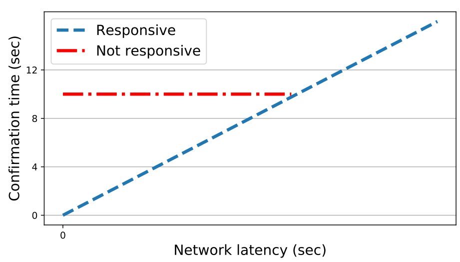


Figure 4: A qualitative comparison of the confirmation time behaviour of parameterized protocols and parameterless ones. Confirmation times in the latter case are fast when the network is smooth and speedy, whereas in the former confirmation time is still limited by the constant hardcoded worst-case bound. Additionally, when the bound of a parameterized protocol is violated, transactions may not be confirmed safely, whereas parameterless protocols adapt to the anomaly and allow confirming transactions more slowly than usual, yet safely.

network, and, in particular, the network remains fully operational, yet slow, during periods of network anomaly.

A second implication of parameterlessness is added security: KNIGHT enjoys a stronger security than existing permissionless protocols, as network hiccups do not interrupt consensus, because they do not violate assumptions necessary for its proper operation.

1.3 Partial Synchrony

Traditionally, a consensus protocol is said to be *partially synchronous* if an upper bound on the network latency exists but is unknown to the protocol [6]. However, proof-of-work consensus introduces some ambiguity into this classification, as it decouples the *transaction ordering* protocol from the *finality protocol*: The core of consensus is the transaction ordering rule (e.g., Bitcoin's longest chain rule, KNIGHT's DAG ordering). This is the canonical algorithm which defines the system, and which all participants run in the same way, including adversarial nodes—individual interpretations of the ledger which differ from the canonical procedure are pointless. In contrast, transaction finality (e.g., the confirmation count in Bitcoin) is a non-binding procedure which each client or user configures and calculates locally according to their own beliefs and needs, and bears the consequence. E.g., a Bitcoin user who believes that malicious miners possess currently less than 33% of the hashrate will confirm transactions faster than one who believes the bound to be 49%, and a 34% attacker will harm the former but not the latter. Another example is SPECTRE, wherein the user needs to additionally specify her belief on the latency bound via a parameter which is configured individually, and which is inconsequential – and, in fact, not-communicated – to the rest of the network.

For these considerations, we believe that a consensus system whose transaction-ordering rule is agnostic to latency should be referred to as partially synchronous and parameterless, interchangeably, even if transaction-finality depends on a latency bound (configured by the user locally). We leave the question on terminology for the Distributed Systems academic community to decide on, and in this paper use the term "parameterless" to describe KNIGHT's operation mode.

1.4 Responsiveness

In the lack of an *a priori* latency assumption, confirmation times in a parameterless consensus system correspond to the real network latency. However, "real latency" has two profoundly different interpretations: the observable latency in the network, and the worst case latency that an attacker may cause. Indeed, a capable attacker may allow – or even assist – the network to operate smoothly, selectively, and disrupt it during a later stage of the attack. A protocol that has the strong property of confirming transactions according to the observable latency is called *responsive* [16].

Despite being parameterless, KNIGHT is *not* responsive in this sense of performing tightly with the current observable latency, rather is responsive in the weaker sense of performing tightly with the current maximum latency causable by an adversary. Indeed, in KNIGHT, it is not enough for the client to set a local bound on the observable latency, rather the bound should reflect the maximum latency that may be caused by the attacker. That is, even if messages currently propagate fully within 1 or 2 seconds, if an attacker may disrupt the network and cause messages to take up to 30 seconds to go through, D should be set by the client to 30 seconds.

This limitation of KNIGHT is unavoidable, since no parameterless protocol with 50% byzantine tolerance threshold can be responsive [16]:

Theorem 14 (Responsive protocols cannot tolerate 1/3 corruption) [Pass and Shi]. No secure permissionless consensus protocol that is also responsive can tolerate 1/3 or more corruption.

1.5 Consensus protocols, principal categories

Consensus protocols are generally classified and compared according to the following aspects:

- What are the assumptions made by the protocol on the underlying network and behaviour of nodes. The stronger the assumptions the weaker the protocol.
- When its assumptions are preserved, how does it perform, specifically, how fast is consensus reached.
- When its assumptions are violated, does the protocol recover, and how fast it recovers. A protocol guaranteed to recover from past failures is called self-stabilizing [5].
- If the underlying system is used to settle a live queue of transactions, we also ask: How many transactions can the protocol serve, i.e., what constraint on the transaction throughput the protocol imposes or its assumptions require.

Through these categories we now survey, with some brevity, KNIGHT 's main properties:

1.5.1 Model assumptions. KNIGHT 's fault model is the byzantine setup, which allows the attacker to deviate arbitrarily from the protocol's rules. We follow the proof-of-work model which assumes a computationally bounded attacker which possesses less than 50% of the computational power in the network. This assumption is considered to be weaker (hence more secure) compared to traditional permissioned setups which require *a priori* knowledge of participating nodes, and compared to proof-of-stake which typically requires a fixed and identifiable set of nodes at the beginning of each epoch.

The attacker is not assumed to suffer any communication delays in its incoming or outgoing links, and may further disrupt honest nodes' communication by delaying messages between them for up to D seconds; however, the D is not known to the protocol. Conversely, the attacker is also capable of speeding up communication between honest nodes down to no-latency; such manipulations are specifically relevant to and challenging in the context of KNIGHT.

1.5.2 Confirmation times (asymptotic). The parameterlessess of the protocol is tightly related to its speed of confirmation: Transaction confirmation times are a function of the actual latency in the network (Subsection 1.3 contains an important reservation of this statement in our context).

Confirmation times are commonly discussed in two modes – optimistic performance and pessimistic performance. The former accounts to the scenario where all participating nodes behave properly, and there is no *visible* attack. In this optimistic scenario, KNIGHT confirms transactions in $O\left(\left(\frac{\ln(1/\epsilon)}{\lambda} + D\right)/(1-2\alpha) + D^2\lambda\right)$ seconds, where λ is the block creation rate in units of blocks/sec (adjusted via a "difficulty adjustment" algorithm adapted from Bitcoin [13]), D is an upper bound on the recent delay in the network, $0 \leq \alpha < 1/2$ is the attacker's size, and $0 \leq \epsilon < 1$ is the required confidence. As in any proof-of-work protocol, the parameters α and ϵ are set by the client. Uniquely to KNIGHT (and to SPECTRE [19]), the parameter D too is set by the client—an underestimation by the client will lead to her premature acceptance of transactions, whereas an overestimation will cause her to wait more time than necessary before confirming.

In the pessimistic scenario, a visible manipulation of the DAG is ongoing, and confirmation times are significantly slowed down. Analyzing the convergence time in this case in a tight manner is intractable, and we are thus left with an exponential bound on confirmation times: $O\left(\frac{1}{\lambda}(\exp(c \cdot D\lambda/(1-2\alpha)) + \ln(1/\epsilon)/(1-2\alpha))\right)$ seconds. We emphasize, however, that this bound is far from tight, assumes an unrealistically strong attacker, and furthermore payments of honest users can still be confirmed in quadratic time as in the optimistic case. Indeed, as long as the user did not publish a *visible* conflict (aka *double spend*), her transaction is commutative with other recent transactions in the DAG, hence the receiving client may confirm it despite the ordering still converging.

1.5.3 Self stabilization. Similarly to NC and other proof-of-work consensus protocols, KNIGHT is self-stabilizing: If the 50% threshold was violated at some point in the past, KNIGHT recovers and transactions may be confirmed safely once the conditions are met; the recovery time is linear in the length of the violation phase. Similarly, the latency parameter D which is set by each client locally should correspond to the recent delay in the network, and need not account for the worst case historical latency. Contrast these properties to proof-of-stake protocols, which rely heavily on *finality*, and which fail therefore to recover from historical catastrophes.

1.5.4 Throughput. In contrast to NC, and similar to other DAG-based consensus protocols, KNIGHT remains secure under arbitrarily high throughput configurations—the block rate, and the block size, should be constrained only according to the capacity of nodes' hardware and that of the network's backbone.

All in all, in this work we propose a novel proof-of-work based parameterless consensus protocol. As far as we are aware, KNIGHT is the first proof-of-work protocol to solve consensus under the parameterless model; the only other protocol to operate under this model is SPECTRE, which solves a weaker version of the consensus problem ("weak liveness"), and which supports therefore only the use case of payments where transactions of honest users are commutative [19]. KNIGHT is a parameterless evolution of PHANTOM—save some nuances, the two coincide when the delay is constant; when the delay is negligible relative to the block creation rate, the two protocols further coincide with NC.

1.6 Structure of this paper

The remainder of this paper is organized as follows. Section 2 contains the full description of the KNIGHT protocol. Section 3 formalizes the model and the statement of KNIGHT's properties. Section 4 discusses confirmation procedure for clients, and confirmation times. In Section 5 we present implementation details. We conclude with surveying related work in Section 6.

2 THE DAG KNIGHT PROTOCOL

2.1 Preliminaries

The following terminology is used extensively throughout this paper. We follow terminology established by previous works concerning DAG protocols [19, 20].

In a block DAG $G = (C, E)$, C represents blocks and E represents hash references to previous blocks—edges thus point backwards in time. $past\,(B, G)$ denotes the set of blocks reachable from B, and $future\,(B, G)$ the set of blocks from which B is reachable; these blocks were provably created before and after B, correspondingly. $anticone\,(B, G)$ denotes the set of blocks outside $past\,(B, G)$ and $future\,(B, G)$; the time-relation between B and blocks in its anticone cannot be derived explicitly from the DAG topology. See Figure 2. When context is clear, we write $anticone\,(B)$ instead of $anticone\,(B, G)$. We denote by $tips\,(G)$ the set of blocks with in-degree 0, that is, which are not referenced by any other block in the DAG. The system is initialized with some known block $genesis$; if a sub-DAG $G' \subseteq G$ has only one block with out-degree 0, we denote it by $genesis(G')$.

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

2.2 PHANTOM optimization paradigm

The PHANTOM protocol [20] proposed an optimization problem as a generalization of NC (see box in Section 1). The optimization targets the largest k-cluster, for a predetermined fixed parameter k which is a function of the worst case latency in the network. In a k-cluster, each block is connected via the DAG topology to all but at most k blocks. Since honest nodes possess a majority of the hashrate, and since blocks created by honest nodes reference one another, the largest k-cluster contains recent honest blocks with high probability, which suffices to secure the ordering.

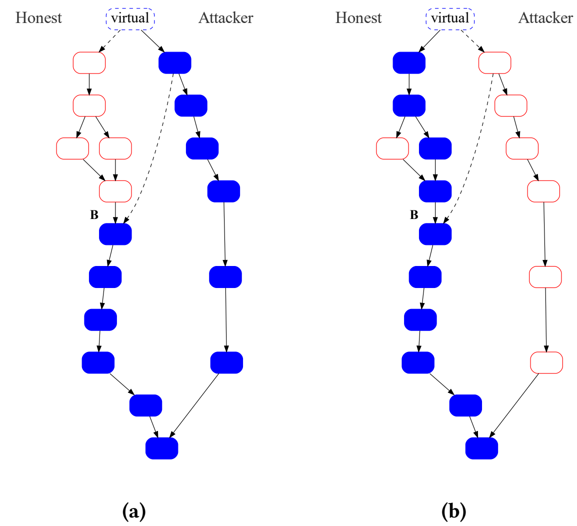


Figure 5: 5a shows a successful freeloading scheme against PHANTOM with $k = 5$. The largest 5-cluster contains (also) attacker blocks which were withheld till now, and excludes honest blocks which were mined correctly above B and published immediately. 5b demonstrates the failure of this scheme against KNIGHT. The protocol recognizes that the largest k-cluster, for $k = 0$, suffices to cover a majority of the DAG, selects this 0-cluster, and excludes the attack blocks.

2.3 KNIGHT optimization paradigm

KNIGHT adds another layer to the optimization problem (as presented in Section 1). Rather than assuming k as an input to the problem, KNIGHT searches for the minimal k such that the largest k-cluster covers at least 50% of the DAG.

This dual minmax optimization (min k, max k-cluster) allows us to tolerate just enough latency and disconnectivity among the selected set of blocks: Intuitively, selecting the cluster of a smaller k would compromise *safety*, exposing the ordering to manipulations by a minority attacker whose blocks do not cover 50% of the graph; selecting the cluster of a larger k would compromise *liveness*, as it would allow adversary blocks to inject themselves into the order even after honest blocks have settled.

Building on this parameterless optimization paradigm, we are able to devise a secure consensus DAG ordering rule that is responsive to the network's actual *adversarial* latency and is not constrained to *a priori* hardcoded bounds on the adversarial latency which require large safety margins and perform suboptimally. We reiterate, however, that KNIGHT is not responsive in the strong sense of performing according to the network's *observable* latency, rather according to the maximum latency that an adversary may cause in the current network; still, under normal Internet conditions, and with sufficient redundancy between peers, this should be in the order of a few seconds at most. In fact, no protocol that is secure against corruption of up to 50% of the nodes can achieve responsiveness in this strong sense, as was proven by Pass and Shi [16].

Figure 5 provides a visual insight into the different behaviour of PHANTOM vs KNIGHT's optimization paradigms. It illustrates a $\sim 35\%$ attacker attempting a "freeloading" manipulation on the respective protocols. Consider the case where PHANTOM was parameterized with $k = 5$, say, and where the honest network enjoys a period of extreme connectivity in the network such that its blocks form a chain (a 0-cluster, in PHANTOM terminology). In a freeloading scheme, the attacker builds her blocks with a certain artificial gap from the rest of the network, 5 in our example. PHANTOM then considers these blocks as part of the largest 5-cluster, and they will precede the second half of the honest chain in the final ordering. KNIGHT, in contrast, is not easily misled—it will recognize that $k = 0$ suffices to cover the majority of the DAG. The same intuition applies generally to any scenario where the network's current (adversarial) latency is smaller than the worst case (adversarial) latency. Moreover, if the network suffers excessive delays due to some anomaly, and PHANTOM's latency bound is violated, transactions may not be confirmed. KNIGHT's operation, in contrast, will remain intact, albeit inevitably slower.

2.4 Vanilla version

Turning KNIGHT's optimization paradigm into a DAG ordering rule seems straightforward:

Algorithm 1 Naïve ordering algorithm

Input: G – the DAG to order
Output: Ordering of G

1: **function** ORDER-DAG(G)
2:     **for** $k = 0, 1 \ldots \infty$ **do**
3:         $S \leftarrow MCS_k$
4:         **if** $|S| \geq \frac{|C|}{2}$ **then**
5:             Order G according to some (deterministic) topological sort that gives precedence to S
6:             **return** the ordered DAG

However, Algorithm 1 is secure only in setups with constant latency and a naïve attacker.[2] We will now describe the full version of KNIGHT, which is secure against both spontaneous and malicious changes in network latency.

2.5 Formal specification

We are now ready to present the holy grail of this work, the DAG-KNIGHT protocol, as specified formally in Algorithm 2. Faithful to its optimization paradigm, the complex of KNIGHT 's procedures (Algorithms 2-6) is designed to recognize the minimal k for which there exists a k-cluster that covers a majority of the DAG. Yet, the choice of cluster entails more than merely finding a large enough cluster, and produces a rather involved algorithm; these complexities will be explained and justified in Subsection 2.6. But first let us provide a bird's-eye view of the algorithm.

2.5.1 Overview. The algorithm receives as input a blockDAG G, and outputs a selected tip of G and an ordering over its blocks, following these steps:

[2]Additionally, line 3 involves solving an NP-hard problem [20].

- Compute the orderings of each tip on its past, and its selected parent, recursively. (Alg. 2)
- Recognize iteratively the next conflict point between the remaining tips $\mathcal{P}$, namely, their latest common chain ancestor g. The induced hierarchy implies that disagreements, or chain-splits, are dealt with from early to recent. (Alg. 2)
- For each conflict point g, decide between competing sets of tips P_i's (which agree with one another on this conflict) by computing each set's *rank* (see below), removing the losers from the set of tips; if needed, run a tie-breaking procedure (Alg. 4).
  This elimination process terminates when a single tip remains ($|\mathcal{P}| = 1$) following the resolution of the most recent conflict point, which is then returned as G's selected tip. (Alg. 2)
- The winning set is that for which at least one representative[3] achieves the lowest rank among all representatives of all sets. (Alg. 3)
- The rank of a block, per a given conflict context, is the minimal k for which the k-cluster *returned by the algorithm* has a subset which *uniformly covers* at least 50% of the DAG. (Alg. 3)
- The k-cluster returned by the algorithm is computed recursively, similarly to the GHOSTDAG cluster-selection (aka colouring) procedure. A subtle condition, represented by the boolean flag $free\_search$, further dictates whether the recursion is called for all parents of the block or only for those which agree with it (on this conflict).[4] (Alg. 5)
- A set of blocks – a subset of the returned cluster – uniformly covers 50% of G if for each block in the set, at least 50% of its future belongs to the set (minus at most $g(k) \in o(k)$). Checking for the existence of a subset which satisfies this property can be done efficiently, by traversing the DAG topologically from tips to genesis, computing each block's "vote" between -1 and 1 (minus 1 represents a block that either does not belong to the original cluster, or violates the coverage condition), and returning genesis' vote.[5] (Alg. 6)

2.5.2 Formal specification. We proceed to present the main procedure of KNIGHT, Algorithm 2, and its subprocedures (Algorithms 3-6). We begin with some formal definitions that the algorithms make use of.

Definition 2. *For a block B, chain-parent (B) is a unique parent of B, set by KNIGHT's chain-selection rule (line 5 in Algorithm 2). The chain of B is defined recursively by* $chain\,(B) := (chain\text{-}parent\,(B)\,,\; chain\text{-}parent\,(chain\text{-}parent\,(B))\,, \ldots, genesis)$.

Observe that *past* (B) fully determines *chain-parent* (B).

Definition 3. *A set of blocks $X \subset G$ is said to* agree in G *if their latest common chain ancestor is a chain-descendant of $g = genesis\,(G)$:* $\exists g' : g \in chain\,(g') \wedge \forall B \in X : g' \in chain\,(B)$.

[3]A representative is any block in the inclusive past of the set of tips which does not belong to the inclusive past of the other sets.
[4]In short, a block may inherit the cluster from a disagreeing parent thereof only if the latter's rank is smaller than former's rank, or if such a condition was met in a previous recursion call.
[5]This procedure borrows from the cascade voting of SPECTRE.

Intuitively, two blocks agree in G if they agree on g's successor in the chain.

Definition 4. *For a set $X \subset tips(G)$ agreeing in G, the set $reps_G(X)$ (*representatives*) is defined by*

$$\left\{x \in \overline{past}(X) \setminus past(tips(G) \setminus X) : x \text{ agrees with } X\right\}.^6$$

Definition 5. *For a block B and chain-ancestor $g \in chain(B)$ s.t. $\exists p_1, p_2 \in parents(B)$ which do not agree over $future(g)$, $rank_{future(g)}(B)$ is defined to be the rank calculated by KNIGHT ordering when recursively executing* Order-DAG $(past(B))$ *and for the iteration of the While loop where g was found (line 12 in Algorithm 2).*

Definition 6. *For $U \subseteq G, d \geq 0$, we say that U is a d-UMC of G (Uniform Majority Coverage), if $genesis(G) \in U$ AND $\forall B \in U$, $future(B) \cap U + d \geq future(B) \cap (G \setminus U)$*

For any non-negative integer k, $g(k) = o(k)$ is a function returning a non-negative integer, used throughout the protocol. We set $g(k) := \lfloor \sqrt{k} \rfloor$. When applied to sets of blocks, max and min [7] operators represent topology relations. That is, if $B = \max G$ then $future(B) \cap G = \emptyset$, and likewise if $B = \min G$ then $past(B) \cap G = \emptyset$.

Algorithm 2 KNIGHT DAG ordering algorithm

Input: G – a block DAG
Output: Selected tip of G, Ordering over G's blocks
1: **function** Order-DAG(G)
2: **if** G is $\{genesis\}$ **then**
3: **return** $genesis$, $[genesis]$
4: **for** $B \in tips(G)$ **do**
5: $chain\text{-}parent(B), order_B \leftarrow$ Order-DAG$(past(B))$
6: $\mathcal{P} \leftarrow tips(G)$
7: **while** $|\mathcal{P}| > 1$ **do**
8: $g \leftarrow$ latest common chain ancestor of all $B \in \mathcal{P}$
9: Partition $\mathcal{P}$ into maximal disjoint sets $\mathcal{P}_1, \ldots, \mathcal{P}_n \subset \mathcal{P}$ s.t. latest common chain ancestor of $\mathcal{P}_i$ is in $future(g)$
10: **for** $\mathcal{P}_i \in \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ **do**
11: $rank_i \leftarrow$ Calculate-Rank $\left(\mathcal{P}_i, future_G(g)\right)$
12: $rank_{G,g} \leftarrow \min_{i \in \{1,\ldots,n\}} rank_i$
13: $\mathcal{P} \leftarrow$ Tie-Breaking$\left(future_G(g), \{\mathcal{P}_i : rank_i = rank_{G,g}\}\right)$
14: $p \leftarrow$ the single element in $\mathcal{P}$
15: **return** p, $\left[order_p \,\|\, p \,\|\, anticone(p)\right]$ ▷ operator $\|$ is sequence concatenation; $anticone(p)$ is iterated in hash-based bottom-up topological order

Algorithm 3 Rank calculation procedure

Input: G – a block DAG, $\mathcal{P}$ – a set of blocks in G (typically $\mathcal{P} \subset tips(G)$)
Output: The rank of $\mathcal{P}$ in G
1: **function** Calculate-Rank($\mathcal{P}, G$)
2: **for** $k = 0, 1 \ldots \infty$ **do**
3: **for** $r \in reps_G(\mathcal{P})$ **do**
4: $C_k(r), \_ \leftarrow$ K-Colouring$(r, G, k, \textbf{false})$
5: **if** UMC-Voting$\left(G \setminus future(r), C_k(r), g(k)\right) > 0$ **then**
6: **return** k

Algorithm 4 Rank tie-breaking procedure

Input: G – a block DAG, $\mathcal{P}_1, \ldots, \mathcal{P}_m \subset tips(G)$
Output: A set of tips $\mathcal{P}_i$ wining the tie-breaking
1: **function** Tie-Breaking($G, \mathcal{P}_1, \ldots, \mathcal{P}_m$)
2: $k \leftarrow$ the mutual rank of $\mathcal{P}_1, \ldots, \mathcal{P}_m$ in G
3: $\mathcal{F}, \_ \leftarrow$ K-Colouring$(virtual(G), G, g(k), \textbf{true})$
4: **for** $\mathcal{P}_i \in \{\mathcal{P}_1, \ldots, \mathcal{P}_m\}$ **do**
5: **for** $k' \in \{\lfloor k/2 \rfloor, \ldots, k\}$ **do**
6: $\_, chain_{i,k'} \leftarrow$ K-Colouring$(virtual(G), G, k', \textbf{false})$ conditioned[8] on $virtual(G)$ agreeing with $\mathcal{P}_i$
7: $C_i \leftarrow \bigcup_{k'} \{B \in \mathcal{F} : anticone(B) \cap chain_{i,k'} > k'\}$
8: $j \leftarrow \operatorname{argmin}_{i \in 1,\ldots,m} \max\{C_i\}$ (break ties according to hash)
9: **return** $\mathcal{P}_j$

2.6 Reviewing the components of KNIGHT

The algorithms presented above are admittedly involved. In this subsection we review their core components. The full version, which will appear online, includes a line by line explanation of the three procedures.

2.6.1 Greedy maximization. To cope with the intractable nature of finding the maximal k-cluster, we take an approach similar to [20] where the NP-hard version was replaced with a greedy procedure, called therein GHOSTDAG. We thus limit the search to extensions of k-clusters of the previous tips of the DAG (K-Colouring, line 7).

2.6.2 Revisiting the Majority condition. Instead of requiring that the selected k-cluster covers a majority of the DAG (equiv., the majority of the future set of the genesis block), we check whether it covers a majority of the future set of each of its member blocks, including genesis; we refer to this property as *uniform majority coverage*, or UMC. Blocks whose future the cluster fails to cover by majority are cast out as outliers, and the procedure counts them outside the cluster. The procedure induces a cascading majority vote (borrowed from [19]) from recent blocks down to the genesis block, and the latter's vote dictates whether the majority cover is satisfactory.

By extending the majority coverage requirement from genesis to any (non-outlier) block in the k-cluster, we recover the "Markovian" nature of the coverage property: Any new honest block "resets" the process by posing an additional challenge to the attacker, namely, to cover the majority of this new block. Indeed, honest miners possess a majority of the hashrate, and blocks of honest miners are referenced by their honest counterparts after at most D seconds, after which honest blocks are expected to win the block race with high probability. To account for these D seconds, we allow the cluster to cover almost a majority—a deficit of $g(k)$ blocks is permitted (line 5 in Calculate-Rank); this relaxed property is called $g(k)$-UMC.

[6] The $past$ operator is used on a set here and reflects the union over $past(B)$ for every block in the set.
[7] As well as argmax, argmin.
[8] Meaning that in this call to K-Colouring, $virtual(G)$ is considered to agree with $\mathcal{P}_i$.

Algorithm 5 k-colouring algorithm

Input: G – a block DAG, C – a block in G, k – a non-negative integer, *free_search* – a Boolean indicating if the search can maximize freely
Output: k-colouring of $past_G(C)$, k-chain of $past_G(C)$

1: **function** K-COLOURING($C, G, k, free\_search$)
2: **if** $past_G(C) = \emptyset$ **then**
3: **return** $\emptyset, \emptyset$
4: $\mathcal{P} \leftarrow \emptyset$
5: **for** $B \in parents(C)$ **do**
6: **if** B agrees with C **then**
7: $blues_B, chain_B \leftarrow$ K-COLOURING$\left(B, past(B) \cap G, k, free_search\right)$
8: $\mathcal{P} \leftarrow \mathcal{P} \cup B$
9: **else if** *free_search* **OR** $k > rank_G(C)$ **then**
10: $blues_B, chain_B \leftarrow$ K-COLOURING$\left(B, past(B) \cap G, k, \textbf{true}\right)$
11: $\mathcal{P} \leftarrow \mathcal{P} \cup B$
12: $B_{\max} \leftarrow \arg\max\{|blues_B| : B \in \mathcal{P}\}$ (break ties according to hash)
13: $blues_G, chain_G \leftarrow blues_{B_{\max}} \cup \{B_{\max}\}, chain_{B_{\max}} \cup \{B_{\max}\}$
14: **for** $B \in anticone(B_{\max}, G)$ **do** in hash-based topological ordering
15: **if** $\left|chain_G \cap anticone(B)\right| \leq k$ **AND** $blues_G \cap anticone(B_{\max}) < k$ **then**
16: $blues_G \leftarrow blues_G \cup \{B\}$
17: **return** $blues_G, chain_G$

Algorithm 6 UMC cascade voting procedure

Input: G – a block DAG, $U \subseteq G$ (typically a k-colouring), e – a non-negative integer representing the deficit threshold
Output: The voting result $vote \in \{-1, 1\}$ of $U \subseteq G$

function UMC-VOTING(G, U, e)
$v \leftarrow \sum_{B \in U}$ UMC-VOTING$\left(future(B), U \cap future(B), e\right)$
return $sign\left(v - \left|G \setminus U\right| + e\right)$ ▷ where $sign(x) := \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases}$

2.6.3 Decouple ordering from colouring. Recall that our assumptions allow for an attacker to control the propagation time of any message in the network up to some (unknown) bound D. It follows that the largest k-cluster, for $k \approx 2 \cdot D \cdot \lambda$ (which bounds the expected size of an honest block's natural anticone) is expected to satisfy the majority coverage property ($g(k)$-UMC). One would expect, therefore, that the following procedure would suffice to secure the ordering: *Find the minimal k for which the largest k-cluster satisfies the $g(k)$-UMC property, and order the DAG according to that cluster.*[9]

Albeit, this approach would undermine the stability of the ordering: If the network's latency changes, spontaneously or maliciously, from $d \ll D$ to D the ordering of the DAG would change retroactively from the largest $k(d)$-cluster to the largest $k(D)$-cluster, undermining the convergence guarantee.

To cope with this challenge, we (re)introduce the notion of a main chain, and order the DAG according to this chain. We show this chain to be robust even under changes of delays, rendering the ordering robust. The chain is formed as follows: Each block picks as its chain predecessor the block which minimizes its own k, or more accurately, its rank (ORDER-DAG, line 11). That is, we utilize the optimization problem of KNIGHT to select the chain-predecessor of each block rather than to order the entire historical DAG. This decoupling allows the chain to "represent" different k's along its growth, which correspond to the effective latency at the time. For example, if at chain-level 200 the attacker exposed a side-DAG that required increasing k from 5 to 7, the ordering of past blocks would still be dictated by the chain below level 198, say.

This decoupling of cluster-selection from DAG-ordering spawns an intricate design space with different inter-dependencies between cluster-selection and chain-ordering. Interestingly, some natural candidates turn out to be insecure, erring either on over-stability (thereby compromising liveness) or on over-flexibility (compromising safety). We strike a balance between these two necessary objectives by allowing the cluster-selection to deviate from the chain-selection of lower-ranked blocks (K-COLOURING, line 9).

2.6.4 Tie-breaking for recovery. Consider a temporary anomaly ("Poisson burst") in the block creation process which led to an abnormally high rank K. After the network resumes normal operation, we would like to recover the normal rank, denoted $k^\star$, or otherwise liveness would be compromised (the waiting time for liveness depends on a non-diverging upper bound over the rank); we thus must guarantee healthy growth of the $k^\star$-cluster, even when the current rank K is excessively high.

In this context the tie-breaking rule between two chain-tip candidates of the same rank turns out to be crucial. A naïve rule would prefer the larger K-cluster, yet such a design would allow an attacker to keep the network at its current rank and prevent it from recovering towards $k^\star$. Instead, we identify the tip whose cluster utilized the excessive rank latest, and prefer its counterpart (TIE-BREAKING algorithm, line 8). The resulting chain-selection rule forces a tie-preserving attacker to compete on ranks much lower than the current one, and eventually to compete on the natural rank, $k^\star$.

2.6.5 Adaptiveness to long-term delay changes. In the lack of an *a priori* latency bound, a parameterless protocol necessarily performs according to the actual (adversarial) latency, as discussed in Section 1. However, in the context of a consensus protocol that serves a continuous queue of transactions, the latency might change with time. It would then be undesirable if the protocol performs according to the worst case historical latency rather than the recent latency in the network. We formalize this requirement in Section 3. To achieve this property, the protocol defines a *conflict hierarchy*, eliminates iteratively the losing candidates, and selects the final survival as the chain-predecessor. This logic is implemented in the While loop in ORDER-DAG (line 7).

2.6.6 Representatives and monotonicity. In theory, an attacker may attempt to artificially increase the rank of honest blocks by wasting part of her hashrate to mine blocks that agree with honest blocks but which do not belong to their k-cluster, where k is the current rank of honest blocks. While this scheme can be shown (yet, at the expense of further complication of the analysis) to be overall suboptimal on her part, it does undermine a desired "monotonous" behaviour of the protocol. Consider a DAG G with two tips B and C,

[9] In other words, for $k = 0, 1, \ldots$ run k-GHOSTDAG, and return the first output that satisfies the $g(k)$-UMC property.

and assume that B "won" and is G's selected chain tip. Consider the effect of adding to G a new block E, which references B only. Since E acknowledges B but not C, one would expect the addition of E to only increase the chance of B to win over C, and definitely not to harm it. Alas, if a set of disconnected E's are added to B's future in this manner, they may increase the rank of their part of the DAG, and in particular may flip the choice and lead to the chain going through C. To recover the desired monotonous behaviour (thereby simplifying our security analysis, as a byproduct), we dictate that C competes with B even if B is no longer a tip of G (!) Thus, to win the chain over E, C must enjoy a rank lower than E (the new tip) but also of all blocks in E's past (which are not in C's past), and B in particular; this exemplifies the role of the representative set (Definition 4) used in line 3 of CALCULATE-RANK.

2.7 Runtime complexity

The algorithms specified in the previous article terminate in polynomial time:

Proposition 1. *Algorithm 2 terminates in polynomial time in $|G|$, and returns a tip and an ordering of G.*

PROOF. Observe the following facts:

- The while loop in line 7 decreases the size of $\mathcal{P}$ at each iteration.
- Following line 13 it remains that $\mathcal{P} \neq \emptyset$, thus, after the loop, $|\mathcal{P}| = 1$ (line 14); thus, the return argument is not null, and is an element in $\mathcal{P}$.
- The overall recursion (line 5) terminates since $\forall B \in tips(G)$, $past(B) \subsetneq G$.
- The procedure CALCULATE-RANK terminates in polynomial time, since the output of K-COLOURING $(C, G, k, \cdot)$, for any block $C \in G$, returns a k-UMC[10], for $k = |G|$, since all blocks in G belong to its largest $|G|$-cluster (there are, obviously, much tighter arguments).

□

In a future version of this paper, we will present an equivalent specification that takes as input two blocks and returns their respective ordering. This procedure is useful for certain types of clients (e.g., what are known as "liteclients"), and can be shown to terminate within a constant (in time) number of steps, concretely, in $O(D \cdot \lambda)^2$ steps.

3 MODEL AND FORMAL STATEMENT

We follow the prevalent models for a proof-of-work governed network [13, 15] and its extensions to the block DAG framework [11, 19, 20]. A network of nodes (or *miners*) is denoted $\mathcal{N}$, each node u maintaining a replica of the DAG observable to it G_t^u. The set $\mathcal{H}$ denotes nodes that follow the mining protocol, which dictates that every new block references all tips of the DAG observable to its miner at its creation, and is broadcast by it immediately to the network. The attacker deviates arbitrarily from the mining protocol, and can further accelerate or delay messages from or to honest nodes up to D_t seconds (D_t depends on time since network conditions and connectivity might change with time); we denote by $D_{\max}$, or simply D, the maximal D_t across $t \in [0, \infty)$. Importantly, $D = D_{\max}$ is a function of the block size limit denoted *block_size_limit* (KB), since large messages take longer to propagate. For brevity, we ignore this parameter, and regard the block size as fixed. We emphasize that *block_size_limit* can be increased in the same manner than the block rate λ may be increased, as discussed in Subsection 5.1.

The proof-of-work mechanism targets a certain block creation rate of λ blocks per second, kept (roughly) constant via a difficulty adjustment algorithm, similarly to Bitcoin [13]. We denote the proof-of-work protocol by $pow(\lambda)$. Block creation thus follows a Poisson process with parameter λ, and the next block in the network is created by an honest node with probability $1 - \alpha_t$, for some (unknown, potentially dynamic) $0 \leq \alpha_t < \alpha$ (for $t \in [0, \infty)$). If this inequality is guaranteed to hold for some range $t \geq s$, We say that α is an *s-updated* bound over the attacker's computational power; this definition is used below to emphasize a self-stabilizing property which allows to recover from "51% attacks".

The DAG ordering rule *ORD* is an algorithm that takes as input a DAG of blocks and returns a linear ordering over its blocks. In our parameterless model, as in the closely-related partial synchrony model, the algorithm may take no parameters as input arguments (such as D, α, k, etc.). We require that all blocks in the DAG were mined correctly according to $pow(\lambda)$. If block a admits a path to block b in the DAG, a was necessarily created after b. The DAG topology induces therefore a natural partial ordering, and the gist of the ordering rule is to extend this to a full ordering over the DAG.

3.1 Convergence of the ordering

The following definitions adapt and extend the model from PHANTOM:

Property 1. *An ordering rule ORD is said to be:*

- P*arameterless if its only input argument is a block DAG G; all blocks in G must be mined correctly according to the proof-of-work protocol* $pow(\lambda)$.
- $(1 - \alpha)$-convergent, *if* $\forall t > 0$, $\forall u \in \mathcal{H}$ *and* $\forall b \in G_t^u$:

$$\lim_{r \to \infty} risk(b, t, r) = 0,$$

even when a fraction of at most α of the mining power is byzantine; the convergence rate of $risk(\cdot)$ should be in $O(f(D, \lambda, \alpha))$, for some function f, and, in particular, may not grow indefinitely with t.[11]
- Scalable *if there exists a constant $\alpha > 0$ such that it $(1 - \alpha)$-converges for all $\lambda > 0$; the maximal such α is called the* security threshold *of ORD.*
- Self stabilizing *if the security threshold of ORD depends on the t-updated bound over the attacker's computational power.*[12]

[10] As shown in the full proof in Appendix A, UMC-VOTING returns a positive sign if U is a d-UMC.

[11] Growing indefinitely with t would imply that confirmation times are not bounded.

[12] This property, which is satisfied by many proof-of-work consensus protocols, implies that the protocol recovers from periods where the attacker's computational power exceeded the allowed threshold, and specifically from what is known as "51% attacks". In fact, some of these protocols, including KNIGHT, satisfy a stronger property and allow the computational power of the attacker to exceed the bound for some limited time-intervals in the future. Delving into these nuances is outside our scope.

- Adaptive *if the convergence rate of risk* (b, t, r) *depends on the recent delay rather than the historical delay; formally, if it is in* $O\left(\max_{s\geq t}\left((g(s-t,\alpha)\cdot f(D_s,\lambda,\alpha)\right)\right)$*. The function* g *represents the "memory" of the process, i.e., how far into the* past *current values of* D (D_s) *impact convergence.*

Here, *risk* (b, t, r) is the probability that the ordering between b and any other block c changes between time t and $t + r$ [20].

3.2 Formal statement

We are finally ready to formally state the achievement of the KNIGHT protocol:

Theorem 2. *KNIGHT's ordering rule (Algorithm 2) is parameterless, scalable, self-stabilizing, and adaptive.*

To the best of our knowledge, KNIGHT is the first proof-of-work based protocol to satisfy all of these properties. For some comparisons: NC is not scalable, since its security threshold deteriorates as the block creation rate λ grows; PHANTOM is not parameterless, since its ordering rule takes as input k, corresponding to the network's worst case latency, and for the same reason it is not adaptive.[13] SPECTRE does not guarantee convergence altogether [19].

In Section 4 we will further shed light on the convergence rate of KNIGHT, specifically, on the order of the functions f and g. In Appendix A we will provide a rigorous proof of Theorem 2.

4 CONFIRMATION TIMES

As common in proof-of-work protocols, the procedure for determining the robustness of the ordering – i.e., evaluating the function *risk* – is done by the client locally, outside the context of consensus. The performance of the protocol in terms of speed is captured by the convergence rate of *risk*. This metric should arguably be dissected into two modes, optimistic and pessimistic. In the former scenario, all participating nodes (miners) seem to behave properly, and in particular there is no visible split in the DAG; formally: all blocks agree on and amplify the entire chain selection, save perhaps a constant-size suffix. In this optimistic scenario, KNIGHT performs very fast, and transactions may be safely confirmed after at most $O\left((\ln(1/\epsilon)+D\cdot\lambda)/(1-2\alpha)+(D\cdot\lambda)^2\right)$ steps, or $O\left(\left(\frac{\ln(1/\epsilon)}{\lambda}+D\right)/(1-2\alpha)+D^2\cdot\lambda\right)$ seconds. In terms of Definition 1, the latter expression describes the asymptotic behaviour of the function f. The function g defined therein can be shown to decay exponentially fast in its argument, implying that confirmation times are highly dependent on the recent worst-case latency in the network, and are insensitive to past or future network hiccups.

In the pessimistic case, where an attacker continuously publishes late blocks and thereby slows down chain solidification, our bounds over confirmation times present an order-of-magnitude slow down: $O(\exp(c\cdot D\cdot\lambda/(1-2\cdot\alpha))+\ln(1/\epsilon)/(1-2\alpha)))$ steps. This describes the asymptotic behaviour of f in the pessimistic scenario (the behaviour of g remains the same). We stress that these bounds are far from tight—they result from the intractability of analyzing the chain solidification under the most sophisticated attack, and further grant the attacker unrealistic communication capabilities. To overcome the intractability, and inspired by a technique from PHANTOM paper, our analysis waits for a rare event in which the honest network mined $Z\cdot D\cdot\lambda$ consecutive blocks in a chain, for some predetermined constant Z. This event is guaranteed to happen within a constant number of steps. While this condition is an overkill, relaxing it and tightening the confirmation times is a complex task, and we defer it to future work.

Notwithstanding, an attacker cannot slow down the confirmation times of regular transactions, even if it carries out a visible attack. As long as the user did not publish an explicit visible conflict to her transaction, its receiver will be able to accept it in the same order-of-magnitude as in the optimistic scenario. Indeed, in this case, the ordering between the published transaction and other transactions would be commutative, and thus the pending chain solidification would be inconsequential to this transaction. Admittedly, in the case of trading against a smart contract, this commutative property might not hold.[14]

The reader may find the comparison between asymptotic confirmation times in KNIGHT and other proof-of-work protocols in Table 1 insightful. Among the protocols under comparison, NC is the fastest to converge under visible (liveness) attacks, yet it converges only for the range $\alpha\in(0,1/(1+D\cdot\lambda))$ [21]. SPECTRE is the fastest to converge under no visible attacks, it converges according to the current (adversarial) latency D_t, and does so slightly faster than KNIGHT does. KNIGHT, in turn, converges in the invisible and visible attack cases, and does so corresponding to D_t as well, in contrast to PHANTOM which converges in terms of $D_{\max}=D$ only. In Section 6 we will survey additional protocols.

Finally, we note that confirmation time analysis of KNIGHT can be tightened significantly when restricted to the attacker range $\alpha<1/3$. We defer this improvement to future work.

5 IMPLEMENTATION DETAILS

An implementation of Algorithm 2 and its subprocedures will be made available online.

5.1 Block size limit

So far we treated synchronous protocols as assuming a bound on latency D. In fact, increasing D and decreasing λ by the same multiplicative factor has no effect and could be regarded as mere change in units. Thus, in truth, the latency assumption takes the form of a bound over $D\cdot\lambda$.

Recall that D depends on the size of messages *block_size_limit* (Section 3). Thus, increasing the block size would have a similar effect to that of increasing the block rate λ.[15] Consequently, in the same manner in which scalable protocols (Definition 1) remain secure under any λ, they remain secure under any block size *block_size_limit*. In the following subsection we discuss whether scalable parameterless protocols, such as KNIGHT, need to limit λ or *block_size_limit*.

[13] Indeed, any latency-parameterized protocol would not be adaptive. However, one may conceive a parameterless protocol that is not adaptive.

[14] These scenarios correspond, essentially, to the consensus properties *safety*, *liveness*, and *weak liveness*, the latter defined in [19].

[15] Notwithstanding, the function $D(block\_size\_limit)$ is nonhomogeneous.

	Visible attack	No visible attack
NC	$O\left(\frac{(\ln(1/\epsilon)+D_t\lambda)}{\max\left\{0,\frac{1-\alpha}{1+D_t\cdot\lambda}-\alpha\right\}}\right)$	(same as the visible attack case, asymptotically)
PHANTOM	$O\left(\exp\left(c_1\frac{D_{\max}\lambda}{1-2\alpha}\right)+\frac{\ln(1/\epsilon)}{1-2\alpha}\right)$	$O\left(\frac{\ln(1/\epsilon)+D_{\max}\lambda}{1-2\alpha}\right)$
SPECTRE	∞	$O\left(\frac{\ln(1/\epsilon)+D_t\lambda}{1-2\alpha}\right)$
KNIGHT	$O\left(\exp\left(c_2\frac{D_t\lambda}{1-2\alpha}\right)+\frac{\ln(1/\epsilon)}{1-2\alpha}\right)$	$O\left(\frac{\ln(1/\epsilon)+D_t\lambda}{1-2\alpha}+(D_t\lambda)^2\right)$

Table 1: A comparison of the convergence rates of different proof-of-work protocols, in terms of time-steps (equiv., number of blocks), in the presence of a visible ongoing liveness attack (left column) and when no such attack is carried visibly (right column). $D_{\max}$ denotes an *a priori* upper bound on the worst case latency, whereas D_t denotes an upper bound on the *current* latency (including possible delays by an adversary). To get expected confirmation times in seconds, multiply each expression by the expected block interval λ^{-1}.

5.2 Difficulty Adjustment Algorithm (DAA)

NC and other proof-of-work protocols employ a DAA that increases the difficulty-target of creating new blocks when the computational power contributed to block creation (aka hashrate) increases, and vice versa when it decreases; refer to [8] for a formal treatment. It is common to ascribe the Sybil-resiliency of the system to this mechanism. However, in truth, proof-of-work suffices to protect against Sybil-nodes even without any DAA. In fact, even if nodes were free to choose the difficulty of their own blocks, one could devise a secure consensus protocol by granting each block a weight, or "voting power", in proportion to its difficulty. Instead, the motivation for DAA is threefold:

- Existing protocols operate in the synchronous setup which assumes an *a priori* bound over the number of blocks created per one unit of delay, i.e., $D \cdot \lambda$. For instance, NC assumes $D \cdot \lambda \ll 1$, and PHANTOM assumes $D \cdot \lambda \ll k+1$. To preserve these bounds and keep the protocol secure, λ cannot increase indefinitely, and must be regulated by the protocol.
- DoS prevention: The capacity of the network and of nodes is limited. The DAA throttles the block creation rate and ensures that the maximum capacity is not exceeded.
- Some application considerations necessitate access to absolute time, such as the regulation of minting, or timelocks. These applications use the block count as a proxy for absolute time.

The first consideration above is irrelevant to KNIGHT, which can cope with dynamic D and λ (and $D \cdot \lambda$). While KNIGHT still requires DAA for the latter considerations – particularly DoS prevention – it could be satisfied perhaps with relaxed versions of DAA. we hope that this discussion spurs new ideas for proof-of-work system designs in the parameterless setup.

6 RELATED WORK

We conclude this paper with a survey of related work. DAG-based protocols have been mentioned extensively throughout the paper, see for example Table 1. Additional relevant protocols include GHOST, which is an alternative chain-selection rule to NC's longest chain, and which performs similarly (in qualitative terms) to NC [9, 10].

Thunderella [17] is a permissionless protocol that is responsive in the strong sense of performing according to the network's actual latency; it requires a super majority of 75% to be honest for this optimistic mode (compared to KNIGHT's 51% majority), as well as the pre-selection of a special "accelerator" node, which compromises the permissionless property of the system. The works in [3, 7, 22] maintain k parallel NC chains, where each block is assigned in random to one of these chains. The ordering rule must then specify the respective ordering between blocks in different chains. These works operate in the synchronous setup, as they pre-assume k so as to ensure that each chain grows with negligible latency; conceptually, as observed by [20], these protocols require $D \cdot \lambda/(k+1) \ll 1$. Prism [3] claim a confirmation time of $O(\max(c1(\alpha) \cdot D, c2(\alpha) \cdot B_v \cdot \ln(1/\epsilon)))$ seconds; here, B_v/C effectively represents the number of blocks per second (λ, in our work). Importantly, in the above term D stands as a function that depends only on network latency and does not depend on the block message size (denoted B_v, and in our work *block_size_limit*). We find this claim questionable, and argue that if indeed D does not depend on *block_size_limit*, then "proposer blocks" and "voter blocks" in Prism do not in fact attest that "transaction blocks" referenced by them have been fully published, which opens up data availability attacks. The number of chains in Prism further depends on the parameter ϵ.

The work in [18] proposes a series of protocols, Slush, Snowflake, and Snowball, which use a network sampling technique to resolve conflicts between nodes. The paper claims very fast confirmation times (1.35 seconds). Yet, these protocols operate in the synchronous model (see Section 2, "Achieving Liveness"), and thus confirmation times in the pessimistic case are not responsive to the network's latency. The protocols are further limited to a fixed confidence parameter ϵ (see e.g. Subsection 3.2 therein), similarly to Prism. Finally, this line of work builds on novel assumptions on nodes' ability to sample the network.

Our work was motivated by Pass and Shi's impossibility result regarding responsive consensus protocols [16], which is an adaptation of the classic 34% byzantine threshold bound on partially synchronous protocols by Dwork et al. [6] to permissionless settings. To circumvent this impossibility, we focused on a relaxed property that aims to be responsive to the maximal latency causable by an adversary, rather than to the observable latency; accordingly, Theorem 2 does not state KNIGHT as being responsive. KNIGHT respects the bound of [6, 16] in that it is responsive to the current worst-case adversarial latency (Δ, in their model) but not to the actual observable one (δ therein). Indeed, the impossibility result (Section 9.2 in [16]) relies directly on the attacker increasing the delay from δ to Δ after transactions have been confirmed. In our model, however, transaction confirmation times depend on Δ (D_t, in our notation).

For discussion of tighter transaction confirmation policies, which employ absolute time in addition to the ledger state, see [14]. The results therein apply, qualitatively, to KNIGHT as well.

The achievement of KNIGHT is made possible by the decoupling of the canonical protocol dictating the ordering over all transactions, and the client protocol for estimating its finality (Subsection 1.4). A similar decoupling was previously proposed in the Flexible BFT paper [12]. That work operates in the traditional permissioned setup, and is still restricted by the 34% threshold for byzantine resilience of partially synchronous protocols, by Dwork et al. [6]. To reason about the fundamental difference in paradigm, observe that the consensus participants in Flexible BFT are not agnostic to the finality of transactions—they reach eventual agreement on finality. In contrast, proof-of-work miners running KNIGHT need not reach any agreement on finality of transactions, and are in fact not required to interpret the state whatsoever. We believe that this agnosticism towards the state is necessary when designing a 50% BFT parameterless consensus protocol.

REFERENCES

[1] http://bitcoinfibre.org/stats.html.
[2] http://kgi.kaspad.net/.
[3] Vivek Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. Prism: Deconstructing the blockchain to approach physical limits. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 585–602, 2019.
[4] Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, Patrick McCorry, Sarah Meiklejohn, and George Danezis. Sok: Consensus in the age of blockchains. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, pages 183–198, 2019.
[5] Edsger W Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974.
[6] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988.
[7] Matthias Fitzi, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Parallel chains: Improving throughput and latency of blockchain protocols via parallel composition. *IACR Cryptology ePrint Archive*, 2018:1119, 2018.
[8] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol with chains of variable difficulty. In *Annual International Cryptology Conference*, pages 291–323. Springer, 2017.
[9] Aggelos Kiayias and Giorgos Panagiotakos. On trees, chains and fast transactions in the blockchain. Cryptology ePrint Archive, Report 2016/545, 2016.
[10] Lucianna Kiffer, Rajmohan Rajaraman, and Abhi Shelat. A better method to analyze blockchain consistency. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 729–744, 2018.
[11] 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.
[12] Dahlia Malkhi, Kartik Nayak, and Ling Ren. Flexible byzantine fault tolerance. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pages 1041–1053, 2019.
[13] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.
[14] Seb Neumayer, Mayank Varia, and Ittay Eyal. An analysis of acceptance policies for blockchain transactions. *Cryptology ePrint Archive*, 2018.
[15] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. *IACR Cryptology ePrint Archive*, 2016:454, 2016.
[16] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model, 2016.
[17] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 3–33. Springer, 2018.
[18] Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gün Sirer. Scalable and probabilistic leaderless bft consensus through metastability. *arXiv preprint arXiv:1906.08936*, 2019.
[19] Yonatan Sompolinsky, Yoad Lewenberg, and Aviv Zohar. SPECTRE: A fast and scalable cryptocurrency protocol. *IACR Cryptology ePrint Archive*, 2016:1159, 2016.
[20] Yonatan Sompolinsky, Shai Wyborski, and Aviv Zohar. PHANTOM and GHOSTDAG: A scalable generalization of nakamoto consensus. Cryptology ePrint Archive, Report 2018/104, 2018. https://eprint.iacr.org/2018/104.
[21] 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.
[22] Haifeng Yu, Ivica Nikolic, Ruomu Hou, and Prateek Saxena. Ohie: Blockchain scaling made simple. *arXiv preprint arXiv:1811.12628*, 2018.

A SECURITY PROOF

A.1 Definitions and Notation

Definition 7. *G_t is the oracle DAG at time t. Similarly, for block B mined at time (B), we abuse notation and define $G_B := G_{time(B)}$.*

Definition 8. *$\mathcal{F}_k(B, G)$ is the maximization-free k-cluster of block B as calculated by the call K-Colouring $(B, G, k, \textbf{true})$ in Alg. 5.*

Definition 9. *$C_k(B, G)$ is the k-cluster of block B as calculated by K-Colouring $(B, G, k, \textbf{false})$ in Alg. 5. We use $C(B)$ to denote $C_k(B, G)$ for the special case $k = k^\star$. When the context is clear we abbreviate and write simply $C_k(B)$.*

Definition 10. *k-chain (B, G) is the chain of k-maximizing blocks used by K-Colouring $(B, G, k, false)$ in order to compose $C_k(B, G)$. More concretely, the k-chain parent of block B is $B_{\max}$, assigned at line 12 of Alg. 5, and so on, recursively.*

Definition 11. *$\overline{C_k(B)}$ is the complementary set of $C_k(B)$ at time (B). More formally, $\overline{C_k(B)} := G_B \setminus C_k(B)$*

Definition 12. *The merge set of block x is defined by $mergeset(x) := past(x) \setminus past(chain\text{-}parent(x))$. We say block x is* merging *block y if $y \in mergeset(x)$.*

The set of blocks mined after and before block x (in absolute time, as seen by an external oracle), are denoted *before* (x) and *after* (x) respectively. We use subscript notation $X_{x\rangle}, X_{\langle x}$ and $X_{\langle x,y\rangle}$ over a set of blocks X, to indicate $X \setminus after(x)$, $X \cap after(x)$ and $X \cap after(x) \setminus after(y)$ respectively.

We notate $x \rightrightarrows y$ if $y = chain\text{-}parent(x)$. Similarly, we use $x \rightrightarrows_k y$ if y is the k-chain parent of x.

A.2 K-cluster Combinatorics

Unlike the GHOSTDAG k-colouring algorithm (Algorithm 1 in [20]), the K-Colouring procedure in Algorithm 5 uses weaker colouring rules which give precedence to past and current *chain* blocks (line 15 therein). Nonetheless, as we show below, for some larger $K > k$ the resulting cluster is still a valid K-cluster. Denote $K(k) := (2k + 1)(k + 1)$.

Lemma 1. *For any block B and DAG G, the cluster C returned by K-Colouring$(B, G, k, \cdot)$ is a $K(k)$-cluster.*

Proof. The first condition in line 15 implies that a block added to the colouring cannot have more than k chain-blocks in its anticone.

Hence by the time B is coloured and added to C, is holds that $|anticone(B) \cap C| \leq (k+1)(k+1)$. This follows from the fact that each of the k chain blocks prior to the chain-block merging B has merged at most k blocks (from the second condition in line 15).

Likewise, only the next k chain-blocks can colour blocks $\in anticone(B)$, resulting in $k(k+1)$ more blocks. Combined we get $|anticone(B) \cap C| \leq (2k+1)(k+1) = K(k)$ □

Lemma 2. *Let $B_1, B_2, \ldots, B_{n-1}, B_n$ be a sequence of k-chain blocks s.t. $B_{i-1} \rightrightarrows_k B_i$ and let $B \in C_k(B_1) \setminus C_k(B_2)$ s.t. B_n is the maximal element $\in$ k-chain $(B_1) \cap past(B)$; then $C_k(B_1) \setminus C_k(B_n) \leq 4K(k)$.*

Proof. [16] The set $past(B_1) \setminus past(B_n)$ is covered by the anticones of $B, B_2, B_{n-1}, B_n \in C_k(B_1)$, which immediately implies its intersection with $C_k(B_1)$ is $\leq 4K(k)$. To see the covering, assume there exists a block D not in any anticone, so $D \notin anticone(B)$; if $D \in future(B)$ then it must be $\in anticone(B_2)$, otherwise $B \in past(B_2)$; if on the other hand $D \in past(B)$ it cannot be in $future(B_{n-1})$ since that will contradict maximality of B_n, thus it must be $\in past(B_{n-1})$ which implies $D \in anticone(B_n)$, a contradiction. □

Lemma 3. *Let $B_1, \ldots, B_{n-1}, B_n$ be a sequence of k-chain blocks s.t. $B_{i-1} \rightrightarrows_k B_i$ and let $B \in C_k(B_1)$ s.t. B_n is the maximal element $\in$ k-chain $(B_1) \cap past(B)$, then $\big(future(B_n) \setminus future(B)\big) \cap C_k(B_1) \leq 2K(k)$.*

Proof. The set $future(B_n) \setminus future(B)$ is covered by the anticones of $B, B_{n-1} \in C_k(B_1)$, which immediately implies its intersection with $C_k(B_1)$ is $\leq 2K(k)$. To see the covering, assume there exists a block in this set s.t. $D \notin anticone(B), \notin anticone(B_{n-1})$; so $D \in past(B)$ hence it cannot be in $future(B_{n-1})$ since that will contradict maximality of B_n, so it must be $\in past(B_{n-1})$ which implies $D \notin future(B_n)$, a contradiction. □

A.3 Main theorem

We are now ready to prove our main result. For the reader's convenience, we first restate the claim:

Theorem 2. *KNIGHT's ordering rule (Algorithm 2) is parameterless, scalable, self-stabilizing, and adaptive.*

A.4 Proof

We fix an arbitrary honest node $u \in honest$ and assume its point of view. We thus abbreviate $virtual_t$ to represent the virtual block of node u at time t. We additionally regard the pov of a hypothetical oracle node that sees all blocks immediately upon their creation, including the attacker; in fact, the omnipotent attacker in our model enjoys the same pov as the oracle node.

Below, we will skip proofs of the more straightforward claims. We will close this gap in a future version of this paper.

A.4.1 Chain growth. We begin by analyzing the growth rate of the maximization-free colouring $\mathcal{F}_k(virtual_t)$ and show that there exists a "natural" k for which this colouring yields > 50% growth. This "free-search" algorithm can be seen as a *modified* GHOSTDAG [20], where by weakening the colouring rules we were able to strengthen the growth-rate proven for GHOSTDAG from achieving a relative majority (over the attacker), to achieving an absolute majority.

Claim 1. *There exists $k^{natural}$, depending only on D, α and λ, s.t. the expected growth of $\mathcal{F}_{k^{natural}}(virtual_t)$ is strictly larger than 0.5; that is,*

$$\mathbb{E}\big(\left|\mathcal{F}_{k^{natural}}(virtual_{t+r})\right| - \left|\mathcal{F}_{k^\star}(virtual_t)\right|\big) > 0.5r\lambda.$$

We now use the maximization-free expected growth-rate, and utilize the Tie-Breaking algorithm, to show that larger-than 50% growth-rate is achieved also for the non-free colouring, albeit with a larger k parameter.

Proposition 3. *There exists $k^\star$, depending only on D, α and λ s.t. the expected growth of $C_{k^\star}(virtual_t)$ is strictly larger than 0.5; that is,*

$$\mathbb{E}\big(\left|C_{k^\star}(virtual_{t+r})\right| - \left|C_{k^\star}(virtual_t)\right|\big) > 0.5r\lambda.$$

Corollary 2. *If U is d-UMC of G, then UMC-Voting$\big(G, U, d\big) > 0$.*

Claim 3. *UMC-Voting$\big(G_t, C_{k^\star}(virtual_t), g(k^\star)\big)$ has positive value in expectation.*

A.4.2 Liveness collapse.

Definition 13. *The attacker advantage $adv(t)$ at time t as $adv(t) := \max_{B \in C_{k^\star}(virtual_t)} future(B) \cap (G_t \setminus C_{k^\star}(virtual_t)) - future(B) \cap C_{k^\star}(virtual_t)$.*

Lemma 4. *The attacker advantage $adv(t)$ is upper bounded by a stochastic process which admits a stationary distribution with $C_0 = O(k^{natural})$ skew and an exponentially decaying tail.*

Definition 14. *A* burst *event $\mathcal{B}_{t,Z}$ is an honest chain burst of size $C_0 + 3Z$ starting at time t, where Z is a function of $K(k^\star)$.*

Let $\mathcal{B}$ denote the sequence of blocks constituting the burst event $\mathcal{B}_{t,Z}$ and let $\mathcal{B}_i$ be the i'th block from the start of the event. Denote the particular blocks $\mathbf{s} := \mathcal{B}_1, \phi := \mathcal{B}_{C_0}, \mathbf{d} := \mathcal{B}_{C_0+Z}, \mathbf{e} := \mathcal{B}_{C_0+2Z}$. These blocks represent the starting point $\mathbf{s}$ of the event, the *pivot* block ϕ which we claim to be on any future honest chain, the *defeat* block $\mathbf{d}$ representing the point where the attacker is in sufficient deficit, and the end block of the burst, $\mathbf{e}$.

Definition 15. *An honest block-race* win *event $\mathcal{W}_t$ is the event that starting from time t, $\forall s > t, C(virtual_s)_{\langle t} \geq \overline{C(virtual_s)}_{\langle t}$*

Definition 16. *The event-sequence $\mathcal{E}_{t,Z}$ is defined to be the sequence of events (i) $adv(t) \leq C_0$, (ii) followed by a burst $\mathcal{B}_{t,Z}$, (iii) followed by a block-race win $\mathcal{W}_{time(\mathbf{e})}$. Note that all events are independent and have positive probability.*

Definition 17. *$\mathcal{A}$ is the set of non-convinced blocks following the burst event, i.e., the set $\big\{B \in G_{\langle \mathbf{e},\infty\rangle} : \phi \notin chain(B)\big\}$.*

Definition 18. *$\mathcal{H}_c$ is the set of chain blocks of honest node u, starting from block $\mathbf{d}$ of the burst event, i.e., the set $\big\{B \in G_{\langle \mathbf{d},\infty\rangle} : \exists t > time(\mathbf{d}), B \in chain(virtual_t)\big\}$.*

Claim 4. *For block $a \in \mathcal{A}$, denote $a_1 = \min chain(a) \cap after(\mathbf{e})$ and $a_2 = \max chain(a) \cap before(\mathbf{s})$. Then it holds that $\forall p \in \mathcal{B}_{\langle \phi}, p \in anticone(a_1) \cup anticone(a_2)$.*

[16] The proofs of the current and following lemmas are in spirit of PHANTOM's freeloader bound [20].

Proof. Assume there exists such $p \notin anticone(a_1) \cup anticone(a_2)$, then $p \in past(a_1) \cap future(a_2)$, so $p \in chain(a)$, contradicting $a \in \mathcal{A}$. □

Claim 5. *For block $a \in \mathcal{A}$, $C_k(a) \cap \mathcal{B}_{\langle\phi} \le 2K(k)$.*

Proof. Follows from Claim 4 and from the definition of a k-cluster. □

The proof of the claim below assumes that the colouring of $chain(B)$ coincides with k-$chain(B, G)$, for any k and for any sub-DAG G. Following the proof we alleviate this assumption. Additionally, for legibility, the proof does not distinguish explicitly between $k^\star$ and $K(k^\star)$. This merely means that some of the constants such as Z need to be set larger and with respect to $K(k^\star)$ rather than $k^\star$.

Claim 6. (main claim) *Conditioned on the occurrence of event-sequence $\mathcal{E}_{t,Z}$, it holds that for any $h \in \mathcal{H}_c \setminus \mathcal{A}$, and for any $a \in \mathcal{A}$ merging h,*

$$past(a) \cap after(h) \setminus future(h) \ge C(h)_{\langle \mathbf{d}} - \overline{C(h)}_{\langle \mathbf{d}}.$$

Proof. Assume for contradiction the claim is false. We look at the minimal event h, a violating the claim statement, i.e., $a \in \mathcal{A}$ is a merging block of $h \in \mathcal{H}_c \setminus \mathcal{A}$ and $past(a) \cap after(h) \setminus future(h) < C(h)_{\langle \mathbf{d}} - \overline{C(h)}_{\langle \mathbf{d}}$.

Denote g to be the most recent shared chain-ancestor of h and a, i.e., $g = \max chain(h) \cap chain(a)$. We analyze the run of Algorithm 2 for the recursive call where $G = past(a)$ (line 5), and for the iteration of the While loop at which g is obtained (line 8), and reach a contradiction to the algorithm's decision. Throughout the proof and sub-claims, we implicitly use a context DAG $\boldsymbol{C}$ which all sets are intersected with. We set the broader context to be $\boldsymbol{C} = \overline{future(g)} \cap past(a)$, however at some inner arguments we narrow the context further.

From minimality of a we have that $future(h) \cap \mathcal{A} = \emptyset$. To see this, assume otherwise and let $a' = \min future(h) \cap \mathcal{A}$. So $h \in mergeset(a')$ since $h \notin \mathcal{A}, a' \in \mathcal{A}$. Additionally, since $a' \in past(a)$ it holds that $past(a') \subset past(a)$, hence $past(a') \cap after(h) \setminus future(h) \subset past(a) \cap after(h) \setminus future(h) < C(h)_{\langle \mathbf{d}} - \overline{C(h)}_{\langle \mathbf{d}}$, contradicting minimality of a.

We now prove that $\forall q \in anticone(h) \cap \mathcal{A}, rank_{\boldsymbol{C}}(h) < rank_{\boldsymbol{C}}(q)$.

Claim 6.1. *$C(h)$ is a $(4k^\star + 2)$-UMC of $\mathbf{C} \setminus future(h)$.*

Proof. In the following, we narrow the implicit context to be $\boldsymbol{C} \setminus future(h)$.

By definition of a UMC it needs to be shown that *every* block in $C(h)$ has bounded negative score (within the context). More formally, we need to show that for every block $b \in C(h)$, it holds that $future(b) \cap C(h) + 4k^\star + 2 \ge future(b) \setminus C(h)$.

Intuitively, while blocks before and during the start of the burst enjoy the natural advantage of the burst, for blocks following $time(\mathbf{d})$ a more sophisticated argument, using the contradiction hypothesis, is required. We thus begin by proving a tighter result for chain blocks mined after $time(\mathbf{d})$, subsequently using it to prove the bound for all blocks in $C(h)_{\langle \mathbf{d}}$.

Claim 6.1.1. $\forall p \in chain(h)_{\langle \mathbf{d}}, future(p) \cap C(h) + 2k^\star + 2 \ge future(p) \setminus C(h)$.

Proof. Note that $p \in \mathcal{H}_c \setminus \mathcal{A}$ since $p \in chain(h), h \in \mathcal{H}_c \setminus \mathcal{A}$, thus from minimality of h, a we have that $after(p) \setminus future(p) \ge C(p)_{\langle \mathbf{d}} - \overline{C(p)}_{\langle \mathbf{d}}$.

Partition $after(p)$ into the following disjoint sets: $m := future(p) \cap C(h)$, $v := after(p) \setminus future(p)$ and $u := future(p) \setminus C(h)$. Additionally, define $\ell := after(h)$. The following claims show relations regarding these definitions.

Claim 6.1.1.1. $\overline{C(h)} - \overline{C(p)} \ge u + v - \ell - k^\star - 1$.

Proof. Since $p \in chain(h)$ it follows by incrementality of $C(h)$ over $chain(h)$ that $C(p) \subset C(h)$. It also follows by $k^\star$-cluster anticone bound that $\overline{anticone}(p) \cap C(h) \le k^\star + 1$.

We now contrast both expressions $\overline{C(h)} - \overline{C(p)}$ and $u + v - \ell$ with the set $G_{\langle p,h \rangle} \setminus C(h)$ and show that they differ only by $k^\star + 1$.

Observe that $\overline{C(p)} = G_p \setminus C(p) = G_p \setminus C(h) + G_p \cap (C(h) \setminus C(p))$, and that $\overline{C(h)} = G_h \setminus C(h)$. Thus by subtraction we obtain $\overline{C(h)} - \overline{C(p)} = (G_h \setminus G_p) \setminus C(h) - G_p \cap (C(h) \setminus C(p)) = G_{\langle p,h \rangle} \setminus C(h) - (\overline{before}(p) \setminus past(p)) \cap C(h)$.

On the other hand $v + u - \ell = after(p) \setminus future(p) + future(p) \setminus C(h) - after(h) = (after(p) \setminus future(p)) \setminus C(h) + (after(p) \setminus future(p)) \cap C(h) + future(p) \setminus C(h) - after(h) = G_{\langle p,h \rangle} \setminus C(h) + (after(p) \setminus future(p)) \cap C(h)$.

Combining both parts we have $\overline{C(h)} - \overline{C(p)} + (\overline{before}(p) \setminus past(p)) \cap C(h) = G_{\langle p,h \rangle} \setminus C(h) = v + u - \ell - (after(p) \setminus future(p)) \cap C(h)$, thus $\overline{C(h)} - \overline{C(p)} = v + u - \ell - \overline{anticone}(p) \cap C(h) \ge v + u - \ell - k^\star - 1$. □

Claim 6.1.1.2. $m + k^\star + 1 \ge C(h) - C(p)$.

Proof. As shown in the previous claim, $C(p) \subset C(h)$. It follows by elementary set logic that $C(h) - C(p) = C(h) \setminus C(p) = C(h) \setminus past(p) = C(h) \cap \overline{anticone}(p) + C(h) \cap future(p) \le k^\star + 1 + m$; where the last transition is from $k^\star$-cluster anticone bound. □

Using the above definitions and the contradiction hypothesis we have $v \ge C(p)_{\langle \mathbf{d}} - \overline{C(p)}_{\langle \mathbf{d}}$ and $\ell < C(h)_{\langle \mathbf{d}} - \overline{C(h)}_{\langle \mathbf{d}}$. Negating the first inequality and summing the expressions we obtain

$$C(h)_{\langle \mathbf{d}} - C(p)_{\langle \mathbf{d}} > \ell - v + \overline{C(h)}_{\langle \mathbf{d}} - \overline{C(p)}_{\langle \mathbf{d}}.$$

Applying Claims 6.1.1.1, 6.1.1.2 on both sides we get that $m + k^\star + 1 > \ell - v + v + u - \ell - k^\star - 1 = u - k^\star - 1$, which translates to the desired result: $future(p) \cap C(h) + 2k^\star + 2 \ge future(p) \setminus C(h)$.

□

For a non-chain block $b \in C(h)_{\langle \mathbf{d}} \setminus chain(h)$, denote $p = \max chain(h)_{\langle \mathbf{d}} \cap past(b)$. Plugging $B_1 = h, B_n = p, B = b$ into Lemma 3 we get that $future(b) \cap C(h) + 2k^\star \ge future(p) \cap C(h)$. Combining with Claim 6.1.1 we conclude that $future(b) \cap C(h) + 4k^\star + 2 \ge future(p) \cap C(h) + 2k^\star + 2 \ge future(p) \setminus C(h) > future(b) \setminus C(h)$; where that last inequality follows from $future(p) \supset future(b)$.

It remains to prove the bound for blocks before and during the start of the burst. For a block $b \in C(h)_{\mathbf{s}\rangle}$, we have from event-sequence $\mathcal{E}_{t,Z}$ that $adv(\mathbf{s}) \le C_0$, thus by definition $future(b)_{\mathbf{s}\rangle} \cap$

$C(h) + C_0 \geq future(b)_{\mathbf{s}\rangle} \setminus C(h)$. Additionally, by construction of the burst event, $future(b)_{\langle\mathbf{s},\mathbf{d}\rangle} \cap C(h) = C_0 + Z > future(b)_{\langle\mathbf{s},\mathbf{d}\rangle} \setminus C(h) = 0$. Finally, since $after(h) < C(h)_{\langle\mathbf{d}} - \overline{C(h)}_{\langle\mathbf{d}}$, it follows that $future(b)_{\langle\mathbf{d}} \cap C(h) \geq future(b)_{\langle\mathbf{d}} \setminus C(h)$. Summing over all time periods we get that $future(b) \cap C(h) \geq future(b) \setminus C(h)$, as claimed.

For blocks $b \in C(h)_{\langle\mathbf{s},\mathbf{d}\rangle}$, similar arguments hold. □

Claim 6.2. *$\forall q \in anticone(h) \cap \mathcal{A}, \forall k \leq \frac{Z-5k^\star}{4}$, $C_k(q)$ is not a $\frac{Z-5k^\star}{4}$-UMC of* $\mathbf{C} \setminus future(q)$.

Proof. We seek to show the existence of a weak block in $C_k(q)$ which has negative score greater than $\frac{Z-5k^\star}{4}$, thus disobeying the UMC requirement. We show this over a maximal pre-burst block in the intersection $C(h) \cap C_k(q)$.

We begin by showing that the attacker cannot effectively freeload following the burst event. To that end, we show in the following claim that $C_k(q)_{\langle\mathbf{e}}$ is bounded in size by the number of blocks out of $C(h)_{\langle\mathbf{e}}$.

Claim 6.2.1. $C_k(q)_{\langle\mathbf{e}} \leq past(q)_{\langle\mathbf{e}} \setminus C(h)_{\langle\mathbf{e}}$.

Proof. First, in the simple case where $C_k(q)_{\langle\mathbf{e}} \cap C(h)_{\langle\mathbf{e}} = \emptyset$ the proof is immediate since $C_k(q)_{\langle\mathbf{e}} \subseteq past(q)_{\langle\mathbf{e}} \setminus C(h)_{\langle\mathbf{e}}$.

For the more complex case, where $C_k(q)_{\langle\mathbf{e}} \cap C(h)_{\langle\mathbf{e}} \neq \emptyset$, we define a counting process and reach the desired result using the bound $k \leq \frac{Z-5k^\star}{4}$.

Define a sequence of chain blocks $q = q_0, \ldots, q_n \in \mathcal{A}$ in the following way:

- Denote $\Delta_{i-1} := C(h) \cap C_k(q) \cap mergeset(q_{i-1})$
- Given q_{i-1}, if $\Delta_{i-1} \neq \emptyset$, select q_i to be $\max chain(q_{i-1}) \cap past(\Delta_{i-1})$.
- Otherwise if $\Delta_{i-1} = \emptyset$, select q_i to be $\max chain(q_{i-1})$ s.t. $C(h) \cap C_k(q) \cap mergeset(q_i) \neq \emptyset$ if such a block exists, or $\max chain(q_{i-1}) \cap before(\phi)$ otherwise.
- If $q_i \in before(\phi)$, halt the process and set $n = i$.

It is true by construction that $\dot{\bigcup}_{i=1}^{n} past(q_{i-1})_{\langle\mathbf{e}} \setminus past(q_i)_{\langle\mathbf{e}}$ is a partitioning of $past(q)_{\langle\mathbf{e}}$. It thus remains to show that for each partition i, $C_k(q) \cap past(q_{i-1}) \setminus past(q_i) \leq past(q_{i-1}) \setminus past(q_i) \setminus C(h)$.

For the first case where $\Delta_{i-1} \neq \emptyset$, let b be any element of Δ_{i-1}. Plugging $B_1 = q_{i-1}, B_n = q_i, B = b$ into Lemma 2 we get that $C_k(q) \cap past(q_{i-1}) \setminus past(q_i) \leq 4k \leq Z - 5k^\star$. Additionally, it can be shown (from minimality of h, a and from block-race condition) that $past(q_{i-1}) \cap after(b) \setminus future(b) \geq Z - 4k^\star$, thus $past(q_{i-1}) \setminus past(q_i) \setminus C(h) \geq Z - 5k^\star$. Combined, $C_k(q) \cap past(q_{i-1}) \setminus past(q_i) \leq Z - 5k^\star \leq past(q_{i-1}) \setminus past(q_i) \setminus C(h)$, as claimed.

In the second case where $\Delta_{i-1} = \emptyset$, the result is immediate since by construction $C_k(q) \cap past(q_{i-1}) \setminus past(q_i) \subseteq past(q_{i-1}) \setminus past(q_i) \setminus C(h)$.

□

We proceed by using the above to show that $C_k(q)$ has smaller than Z advantage within post-burst blocks.

Claim 6.2.2. $C_k(q)_{\langle\mathbf{e}} < past(a)_{\langle\mathbf{e}} \setminus C_k(q) \setminus future(q) + Z$.

Proof. Recall that $after(h) \setminus future(h) < Z + C(h)_{\langle\mathbf{e}} - \overline{C(h)}_{\langle\mathbf{e}}$. Reorganizing terms we obtain that $after(h) \setminus future(h) + \overline{C(h)}_{\langle\mathbf{e}} < Z + C(h)_{\langle\mathbf{e}}$; noting that by definition $past(a)_{\langle\mathbf{e}} \setminus C(h) \setminus future(h) = \overline{C(h)}_{\langle\mathbf{e}} + after(h) \setminus future(h)$ we derive that $past(a)_{\langle\mathbf{e}} \setminus C(h) \setminus future(h) < C(h)_{\langle\mathbf{e}} + Z$.

Define $m := C(h)_{\langle\mathbf{e}}$, $u := past(a)_{\langle\mathbf{e}} \setminus C(h) \setminus future(h) \setminus future(q)$ and $v := C_k(q)_{\langle\mathbf{e}}$. We get that $u \leq past(a)_{\langle\mathbf{e}} \setminus C(h) \setminus future(h) < C(h)_{\langle\mathbf{e}} + Z = m + Z$. Adding u to both sides we have $2u < m + u + Z \leq past(a)_{\langle\mathbf{e}} \setminus future(q) + Z$.

From Claim 6.2.1 we have that $C_k(q)_{\langle\mathbf{e}} \leq past(q)_{\langle\mathbf{e}} \setminus C(h)_{\langle\mathbf{e}}$. Noting that $v = C_k(q)_{\langle\mathbf{e}} \leq past(q)_{\langle\mathbf{e}} \setminus C(h)_{\langle\mathbf{e}} \subset past(a)_{\langle\mathbf{e}} \setminus C(h) \setminus future(h) \setminus future(q) = u$, we get that $v \leq u$. Thus $2v < past(a)_{\langle\mathbf{e}} \setminus future(q) + Z$. Reorganizing terms and noting that $v \subset past(a)_{\langle\mathbf{e}} \setminus future(q)$, we conclude that $C_k(q)_{\langle\mathbf{e}} < past(a)_{\langle\mathbf{e}} \setminus C_k(q) \setminus future(q) + Z$, as claimed. □

To complete the argument, we determine the attacker's weak block. Let $w := \max C(h)_{\phi\rangle} \cap C_k(q)_{\phi\rangle}$ (this intersection is not empty as it contains g). If $w \in after(\mathbf{s})$, then $w \in \mathcal{B}$, so from maximality and burst structure $future(w)_{\phi\rangle} \cap C_k(q) - future(w)_{\phi\rangle} \setminus C_k(q) \leq 0$. Otherwise, $w \in \overline{before}(\mathbf{s})$. We have from event-sequence $\mathcal{E}_{t,Z}$ that $adv(\mathbf{s}) \leq C_0$, thus by definition $future(w)_{\mathbf{s}\rangle} \cap C(h) + C_0 \geq future(w)_{\mathbf{s}\rangle} \setminus C(h)$. From maximality, we have that $future(w)_{\mathbf{s}\rangle} \cap C(h)$ and $future(w)_{\mathbf{s}\rangle} \cap C_k(q)$ are disjoint, thus $future(w)_{\mathbf{s}\rangle} \setminus C_k(q) + C_0 \geq future(w)_{\mathbf{s}\rangle} \cap C(h) + C_0 \geq future(w)_{\mathbf{s}\rangle} \setminus C(h) \geq future(w)_{\mathbf{s}\rangle} \cap C_k(q)$. Noticing (again, by maximality) that all C_0 blocks of $\mathcal{B}_{\langle\mathbf{s},\phi\rangle}$ are not in $C_k(q)$ we obtain that $future(w)_{\phi\rangle} \cap C_k(q) - future(w)_{\phi\rangle} \setminus C_k(q) \leq C_0 - C_0 = 0$.

Since $q \in \mathcal{A}$ we have from Claim 5 that $C_k(q) \cap \mathcal{B}_{\langle\phi} \leq 2k \leq \frac{Z-5k^\star}{2}$. Noting that the remainder of the burst is not in $C_k(q)$ we get that $future(w)_{\langle\phi,\mathbf{e}\rangle} \cap C_k(q) - future(w)_{\langle\phi,\mathbf{e}\rangle} \setminus C_k(q) \leq -3Z + Z - 5k^\star \leq -2Z$.

Using Claim 6.2.2 and summing over all time periods we get $future(w) \cap C_k(q) - future(w) \setminus C_k(q) < -Z < -\frac{Z-5k^\star}{4}$, as we need. □

To conclude , it remains to set Z large enough s.t. $4k^\star \leq \frac{Z-5k^\star}{4}$ and thus $\forall q \in anticone(h) \cap \mathcal{A}, rank_{past(a)}(h) < rank_{past(a)}(q)$. Recall that $future(h) \cap past(a) \cap \mathcal{A} = \emptyset$, so by definition $t \in reps(\mathcal{P} \setminus \mathcal{A})$, thus $rank_{past(a)}(\mathcal{P} \setminus \mathcal{A}) \leq rank_{past(a)}(h)$. On the other hand, for any $\mathcal{P}_i$ disagreeing with $\mathcal{P} \setminus \mathcal{A}$, it holds that $reps(\mathcal{P}_i) \subseteq anticone(h) \cap \mathcal{A}$, thus $rank_{past(a)}(\mathcal{P}_i) > rank_{past(a)}(h)$, contradicting $a \in \mathcal{A}$, since a must select a chain parent from $\mathcal{P} \setminus \mathcal{A}$. □

As stated earlier, the above proof relies on some simplifying assumptions. We now set out to show how it can be generalized to the actual colouring algorithm. We first deal with the case where the recursive call within K-Colouring has never set *free_search = true*. This means that $\forall a \in \mathcal{A}, k\text{-}chain(a)_{\langle\mathbf{d}} \subset \mathcal{A}$, thus all inequalities in the above proof trivially hold.

The more challenging case is the one where a recursive call to K-Colouring switches to *free_search = true* (line 10), hence by that allowing the attacker to "inherit" the honest colouring. Observe that this can only happen if $k > rank_G(C)$, thus implicitly forcing a rank increase. The following extended argument captures exactly this property:

Claim 7. (generalized main claim) *Conditioned on the occurrence of event-sequence $\mathcal{E}_{t,Z}$, it holds that for any $h \in \mathcal{H}_c \setminus \mathcal{A}$, and for any $a \in \mathcal{A}$ merging h,*

$$\begin{aligned} &past\,(a) \cap after\,(h) \setminus future\,(h) \\ &\geq C\,(h)_{\langle \mathbf{d}} - \overline{C\,(h)}_{\langle \mathbf{d}} \\ &+ \max\left(0, 4k^{\star} - rank_G\,(a)\right) \cdot Z. \end{aligned} \tag{1}$$

A full proof of this claim will appear in a future version of the paper.

To conclude the proof of Theorem 2, the following Corollary shows that, indeed, following event-sequence $\mathcal{E}_{t,Z}$, the attacker can never regain an advantage:

Corollary 8. *Conditioned on the occurrence of event-sequence $\mathcal{E}_{t,Z}$, the attacker cannot reorg below the burst event. More formally, $\forall s \geq time\,(\mathbf{e}), \phi \in chain\,(virtual_s)$.* [17]

Proof. All sets within the current proof are implicitly intersected with *after* $(\mathbf{e})$.

We first observe that at the starting point, i.e., at time $s = time\,(\mathbf{e})$, it holds by construction of the burst event that $\phi \in chain\,(virtual_s)$. Assume for contradiction there exists a minimal time $s > time\,(\mathbf{e})$ s.t. $virtual_s \in \mathcal{A}$.

We will use the properties of the event-sequence (specifically, block race win and the initial burst advantage) to provide an upper bound on G_{s-1} and a lower bound on G_s, and arrive at a contradiction.

From minimality of s, there exists a block $h \in \mathcal{H}_c \setminus \mathcal{A}$ s.t. $virtual_{s-1} \rightrightarrows h$. Additionally, from block-race win we have that $C\,(virtual_{s-1}) \geq \overline{C\,(virtual_{s-1})}$, thus $2C\,(virtual_{s-1}) \geq G_{s-1}$, which leads to $2C\,(h) + 2k^{\star} \geq G_{s-1}$

On the other hand, at time s, let $a \in \mathcal{A}$ be the merging block of h (be it any $a \in chain\,(virtual_s)$ or $virtual_s$ itself), then by Claim 6 it holds that $past\,(a) \cap after\,(h) \setminus future\,(h) \geq Z + C\,(h) - \overline{C\,(h)}$. It follows that $G_s \geq G_{time(h)} + C\,(h) - \overline{C\,(h)} + Z = C\,(h) + \overline{C\,(h)} + C\,(h) - \overline{C\,(h)} + Z = 2C\,(h) + Z$.

Combined, we get that $2C\,(h) + 2k^{\star} \geq G_{s-1} = G_s - 1 \geq 2C\,(h) + Z - 1$, which yields $2k^{\star} \geq Z - 1$, a contradiction. □

[17] Equivalently: $\mathcal{H}_c \cap \mathcal{A} = \emptyset$.