Inclusive Block Chain Protocols

Yoad Lewenberg[1], Yonatan Sompolinsky[1], and Aviv Zohar[1,2]

[1] The School of Engineering and Computer Science,
The Hebrew University of Jerusalem, Israel
[2] Microsoft Research, Herzliya, Israel
{yoadlew,yoni_sompo,avivz}@cs.huji.ac.il

Abstract. Distributed cryptographic protocols such as Bitcoin and Ethereum use a data structure known as the block chain to synchronize a global log of events between nodes in their network. Blocks, which are batches of updates to the log, reference the parent they are extending, and thus form the structure of a chain. Previous research has shown that the mechanics of the block chain and block propagation are constrained: if blocks are created at a high rate compared to their propagation time in the network, many conflicting blocks are created and performance suffers greatly. As a result of the low block creation rate required to keep the system within safe parameters, transactions take long to securely confirm, and their throughput is greatly limited.
We propose an alternative structure to the chain that allows for operation at much higher rates. Our structure consists of a directed acyclic graph of blocks (the block DAG). The DAG structure is created by allowing blocks to reference multiple predecessors, and allows for more "forgiving" transaction acceptance rules that incorporate transactions even from seemingly conflicting blocks. Thus, larger blocks that take longer to propagate can be tolerated by the system, and transaction volumes can be increased.
Another deficiency of block chain protocols is that they favor more connected nodes that spread their blocks faster—fewer of their blocks conflict. We show that with our system the advantage of such highly connected miners is greatly reduced. On the negative side, attackers that attempt to maliciously reverse transactions can try to use the forgiving nature of the DAG structure to lower the costs of their attacks. We provide a security analysis of the protocol and show that such attempts can be easily countered.

1 Introduction

Bitcoin, a decentralized digital currency system [9], uses at its core a distributed data structure known as the block chain—a log containing all transactions conducted with the currency. Several other distributed systems, such as Ethereum, a general distributed applications platform, have extended Bitcoin's functionality, yet still rely on a similar block chain to synchronize information between nodes.

As Bitcoin, Ethereum, and their likes gain wider acceptance, it is expected that pressure to include more data in their blocks will increase as well. Due to

bandwidth constraints, larger blocks propagate through the network less efficiently, and may thus result in suboptimal performance if too many transactions are included. This is mainly due to the uncoordinated creation of blocks by different nodes which results in conflicts. The current protocols dictate that whenever conflicts occur, only a single block is adopted, and the others are discarded.

This paper explores an alternative mechanism for the formation of the block chain that is better suited for such protocols when block sizes are large, or when blocks are created often. Our modification allows the inclusion of transactions from conflicting blocks. We thus create an incentive for nodes to attempt and include *different* transactions, and thereby increase throughput.

Conflicts, and the structure of the block chain The block chain in each protocol is replicated at every node and assists nodes in reaching a consensus on the state of all "accounts". Blocks, which make up the chain, contain an identifier (a cryptographic hash) of their predecessor in the chain, as well as a set of transactions that are consistent according to the state of the ledger represented by the chain they extend. To avoid creating a monopoly on the approval of transactions, all nodes have the ability to create blocks. To create a block, a node (also known as a miner) has to solve a computationally intense proof of work problem (the proof of work computation essentially consists of guessing inputs to a cryptographic hash function which succeeds only probabilistically). Once a block is created, it is distributed to the rest of the network. Blocks may be created by different nodes at roughly the same time, and may thus extend the same parent block. Such blocks may include different subsets of transactions, some possibly conflicting (conflicting transactions are those that move the same money to different destinations – they cannot be allowed to co-occur). The protocol therefore includes a mechanism for choosing which block survives to extend the chain, while the other conflicting ones are effectively ignored. The mechanism used by Bitcoin is this: given several extensions of the current chain, pick the longest chain as the version to adopt. Ethereum on the other hand uses a different selection strategy which is a variant of GHOST [12] (readers unfamiliar with the basic Bitcoin protocol are referred to [9]).

The chain selection rule can be exploited by a malicious node to reverse a payment, an attack known as *double-spend*. The attacker can attempt to build a secret chain of blocks which does not contain the transaction and later, if its chain is long enough, replace the main chain, thereby reversing the payment.

Previous work [6, 12] has shown that with increasing block sizes (or equivalently with increasing block creation rates), more stale (off-chain) blocks are created. This, in turn, leads to several problems: First, the security of the protocol against malicious attacks suffers. Second, increases in block size do not translate to linear increases in throughput (as the contents of off-chain blocks are not included in the ledger). Finally, the situation in which blocks conflict puts smaller less connected miners at a disadvantage: They earn less than their respective share of the rewards, and may be slowly pushed out of the system due to competition with larger miners, a fact which endangers the decentralization of Bitcoin.

The problems mentioned above form barriers to the scalability of block chain protocols. If block sizes are not increased, competition between transactions that attempt to enter the block chain will raise fees to high levels that discourage use of the protocol.

Indeed, Ethereum's adopted chain selection protocol was specifically designed to provide stronger security guarantees exactly in these high throughput settings [13], but other issues such as the skewed reward distribution at high rates, or the loss of throughput due to excluded blocks have not been improved. Our suggested modification aims to provide an additional improvement, and works well with GHOST, with its variant used by Ethereum, with the standard longest-chain protocol, and in fact, with any protocol that selects a "main" chain.[3]

The Block DAG, and inclusive protocols We propose to restructure the block chain into a directed acyclic graph (DAG) structure, that allows transactions from *all* blocks to be included in the log. We achieve this using an "inclusive" rule which selects a main chain from within the DAG, and then selectively incorporates contents of off-chain blocks into the log, provided they do not conflict with previously included content. An important aspect of the Inclusive protocol is that it awards fees of accepted transactions to the creator of the block that contains them—even if the block itself is not part of the main chain. Such payments are granted only if the transaction was not previously included in the chain, and are decreased for blocks that were published too slowly.

Analysis of such strategies is far from simple. We employ several game theoretic tools and consider several solution concepts making different assumptions on the nodes (that they are profit maximizers, cooperative, greedy-myopic, or even paranoid and play safety-level strategies). In all solution concepts one clear trend emerges: nodes play probabilistically to minimize collisions, and do not choose only the highest fee transactions that would fit into their block.

One potential negative aspect of our suggestion is that attackers that try to double-spend may publish the blocks that were generated in failed attempts and still collect fees for these blocks. We show that this strategy, which lowers the costs of double-spend attacks, can be easily mitigated with slightly longer waiting times for final transaction approval, as the costs of an attacker grow significantly with the waiting time.[4] We additionally consider a new attack scenario (which has not been analyzed in previous work) in which an attacker creates a public fork in the chain in order to delay transaction acceptance by nodes.

Another issue that arises as many conflicting blocks are generated by the protocol, is the problem of selfish mining [7], in which miners deviate from Bitcoin's proposed strategy to increase their gains. Inclusive protocols remain susceptible to this form of deviation as well, and do not solve this issue.

To summarize, our main contributions are:

[3] For the sake of brevity, we do not go into the details of GHOST or of its Ethereum-variant, except where specifically relevant.

[4] This is guaranteed only if the attacker has less than 50% of the computational power in the network.

1. We utilize a directed acyclic structure for the block graph in which blocks reference several predecessors to incorporate contents from all blocks into the log (similar structures have already been proposed in the past, but not to include the contents of off-chain blocks).
2. We provide a game theoretic model of the competition for fees between the nodes under the new protocol.
3. We analyze the game under several game theoretic solution concepts and assumptions, and show that in each case nodes randomize transaction selection from a wider range of transactions. This is the key to the improved performance of the protocol.
4. We demonstrate that Inclusive protocols obtain higher throughput, more proportional outcomes that less discriminate smaller, less-connected players, and that they suffer very little in their security in comparison to non-inclusive protocols. We consider both security against double-spend attempts, as well as attackers that are trying to delay transaction acceptance in the network.

2 From Trees to Directed Acyclic Graphs (DAGs)

We now begin to describe our proposed changes to the protocol. We start with a structural change to the blocks that will enable further modifications. In the current Bitcoin protocol, every block points at a single parent (via the parent's hash), and due to natural (or malicious) forks in the network, the blocks form a tree.

We propose, instead, the node creating the block would list *all* childless blocks that it was aware of. Surely, this added information does not hurt; it is simple to trace each of the references and see which one leads, for example, to the longest chain. We thus obtain a directed acyclic graph (DAG) in which each block references a subset of previous blocks. We assume that when block C references B, C's creator knows all of B's predecessors (it can request them). The information that can be extracted from a block's reference list is sufficient to simulate the underlying chain selection rule: we can follow the longest-chain rule, for example, by recursively selecting in each block a single link—the one leading to the longest chain.

The provision of this additional information amounts to a "direct revelation mechanism": Instead of instructing nodes to select the chain they extend, we simply ask them to report all possible choices, and other nodes can simulate their choice, just as they would have made it (the term *direct revelation* is borrowed from economics where it is widely used in mechanism design [10]).

In fact, any chain selection protocol can be applied in this manner, as the references provide all information needed to determine the choice that the block creator would have made when extending the chain. The only issue that needs to be handled is tie breaking (as in the case of conflicting chains of equal length). To do so, we ask nodes to list references to other blocks in some order, which is then used to break ties. Note that nodes are only required to list the childless nodes

in the DAG; there is no need to list other nodes, as they are already reachable by simply following the links.[5]

Formally, we denote by $BDAG$ the set of all directed acyclic block graphs $G = (V, E)$ with vertices V (blocks) and directed edges E, where each $B \in V$ has in addition an order $\prec_B$ over all its outgoing edges. In our setup, an edge goes from a block to its parent, thus childless vertices ("leaves") are those with no *incoming* edges. Graphs in $BDAG$ are required to have a unique maximal vertex, "the genesis block". We further denote by $sub(B, G)$ the subgraph that includes all blocks in G reachable from B.

An underlying chain selection rule F is used to decide on the main chain in the DAG (e.g., longest-chain or GHOST). The rule F is a mapping from block DAGs to block chains such that for any $G \in BDAG$, $F(G)$ is a maximal (i.e., non-extendable) chain in G. The order $\prec_B$ is assumed to agree with F, in the sense that if A is one of B's parents and $A \in F(sub(B, G))$, then A is first in the order $\prec_B$.

2.1 Exploiting the DAG Structure—The Inclusive Protocol

We define Inclusive-F, the "Inclusive" version of the chain selection rule F, which incorporates non-conflicting off-chain transactions into a given blocks accepted transaction set. Intuitively, a block B uses a postorder traversal on the block DAG to form a linear order on all blocks. If two conflicting transactions appear, the one that appeared earlier according to this order is considered to be the one that has taken place (given that all previous transactions it depends on have also occurred). Thus, we use the order on links that blocks provide to define an order on blocks, which we then use to order transactions that appear in those blocks, and finally, we confirm transactions according to this order.

To make the Inclusive algorithm formal, we need to provide a method to decide precisely the set of accepted transactions. Bitcoin transactions are composed of inputs (sources of funds) and outputs (the targets of funds). Outputs are, in turn, spent by inputs that redirect the funds further. We define the consistency of a transaction set, and its maximality as follows:

Definition 1. *Given a set of transactions T, a transaction tx is* consistent with *T if all its inputs are outputs of transactions in T, and no other transaction in T uses them as inputs. We say that T is* consistent*, if every transaction $tx \in T$ is consistent with $T \setminus \{tx\}$.*

Definition 2. *We say that a consistent set of transactions T from a block DAG G is* maximal*, if no other consistent set T' of transactions from G contains T.*

[5] DAGs are already required by GHOST (although for different reasons), and Ethereum's blocks currently reference parent blocks as well as "uncles" (blocks that share the same parent as their parent). Thus, this modification is quite natural.

The algorithm below performs a postorder traversal of the DAG $sub(B, G)$. Along its run it confirms any transaction that is consistent with those accepted thus far. The traversal backtracks if it visits the same block twice.[6]

The algorithm is to be called with arguments Inclusive-$F(G, B, \emptyset)$, initially setting $visited(\cdot)$ as False for all blocks. Its output is the set of transactions it approves.

Algorithm 1. *Inclusive-*$F(G, B, T)$
Input: a DAG G*, a block* B *with pointers to predecessors* $(B_1, ..., B_m)$ *(ordered according to* $\prec_B$*),*[7] *and a set of previously confirmed transactions* T*.*

1. IF *visited(B)* RETURN T
2. SET *visited(B):=True*
3. FOR $i = 1$ TO m*:*
4. $\quad T =$ *Inclusive-*$F(G, B_i, T)$
5. FOR EACH $tx \in B$
6. $\quad$ IF (tx *is consistent with* T) THEN $T = T \cup \{tx\}$
7. RETURN T

We say that B is a *valid* block if at the end of the run on $sub(B, G)$ we have $B \subseteq T$.[8] The algorithm's run extends $\prec_B$ to a linear order on $sub(B, G)$, defined by: $A \prec_B A'$ if Inclusive-$F(G, B, \emptyset)$ visited A before it visited A'. The following proposition states that the algorithm provides consistent and maximal transaction sets:

Proposition 1. *Let* T *be the set returned by Inclusive-*$F(G, B, \emptyset)$*. Then* T *is both consistent and maximal in* $sub(B, G)$*.*

The proof is immediate from the algorithm.

An important property of this protocol is that once a transaction has been approved by some main chain block B of G, it will remain in the approved set of any extending block as long as B remains in G's main chain. This is because transactions confirmed by main chain blocks are first to be included in the accepted transaction sets of future main chain blocks. Since both in longest-chain and GHOST blocks that are buried deep in the main chain become increasingly less likely to be replaced, the same security guarantees hold for transactions included in their Inclusive versions.

Fees and Rewards Each transaction awards a fee to the creator of the first block that included it in the set T. Formally, let A be some block in $sub(B, G)$. Denote by $T(A)$ the set of transactions which block A was the first to contain,

[6] It is important to note that the algorithm below describes a full traversal. More efficient implementations are possible if a previously traversed DAG is merely being updated (with methods similar to the unspent transaction set used in Bitcoin).

[7] If B is the genesis block, which has no predecessors, $m = 0$.

[8] The Inclusive algorithm can also handle blocks that have some of their transactions rejected.

according to the order $\prec_B$. Then (according to B's world view) A's creator is awarded *a fraction* of the fee from every $tx \in T(A)$. Although naïvely we would want to grant A all of $T(A)$'s fees, security objectives cannot always permit it. This is one of the main tradeoffs in the protocol: On the one hand, we wish to award fees to anyone that included a new transaction. This implies that poorly connected miners that were slow to publish their block will still receive rewards. On the other hand, off-chain blocks may also be the result of malicious action, including published blocks from a failed double-spend attack. In this case we would prefer no payoff would be received. We therefore allow for a somewhat tolerant payment mechanism that grants a block A a fraction of the reward which depends on how quickly the block was referenced by the main chain. The analysis that will follow (in Sect. 3) will justify the need for lower payments.

Formally, for any block $A \in G$ define by $pre(A)$ the latest *main chain* block which is reachable from A, and by $post(A)$ the earliest *main chain* block from which A is reachable; if no such block exists, regard $post(A)$ as a "virtual block" with height infinity; if A is in the main chain then $pre(A) = post(A) = A$. Denote $c(A) := post(A).height - pre(A).height$; $c(\cdot)$ is a measure of the delay in a block's publication (with respect to the main chain).

In order to penalize a block according to its gap parameter $c(\cdot)$ we make use of a generic discount function, denoted γ, which satisfies: $\gamma : \mathbb{N} \cup \{0\} \to [0, 1]$, it is weakly decreasing, and $\gamma(0) = 1$. The payment for (the creator of) block A is defined by:

$$\gamma\left(c(A)\right) \cdot \sum_{w \in T(A)} v(w),$$

where $v(w)$ is the fee of transaction w. In other words, A gains only a fraction $\gamma(c(A))$ of its original rewards. By way of illustration, consider the following discount function:

Example 1.

$$\gamma_0(c) = \begin{cases} 1 & 0 \leq c \leq 3 \\ \frac{10-c}{7} & 3 < c < 10 \\ 0 & c \geq 10 \end{cases} \tag{1}$$

γ_0 grants a full reward to blocks which are adequately synchronized with the main chain ($\gamma_0(c) = 1$ for $c \leq 3$), on the one hand, and pays no reward at all to blocks that were left "unseen" by the main chain for too long, on the other hand ($\gamma_0(c) = 0$ for $c \geq 10$); in the mid-range, a block is given some fraction of the transaction rewards ($\gamma_0(c) = \frac{10-c}{7}$ for $3 < c < 10$).

Money Creation In addition to fees, Bitcoin and other cryptocurrencies use the block creation process to create and distribute new coins. Newly minted coins can also be awarded to off-chain blocks in a similar fashion to transaction fees, i.e., in amounts that decrease for blocks that were not quickly included in the main chain. A block's reward can therefore be set as a fraction $\gamma(c(A))$ of the full reward on the chain.[9] As our primary focus is on the choice of transactions

[9] The total reward can be automatically adjusted to maintain a desired rate of money creation by a process similar to the re-targeting done for difficulty adjustments.

to include in the block, we assume for simplicity from this point on, that no money creation takes place (i.e., that money creation has decayed to negligible amounts—as will eventually occur for Bitcoin).

Now that we have defined the Inclusive protocol, we begin to analyze its implications.

3 Security

The original security analysis of Satoshi ([9]), as well as analysis done by others [11, 12], has considered the *probability* of a successful double-spend attack under the regular non-inclusive scheme. An alternative analysis may instead measure the cost of the attack rather than their success probability (both have been analyzed in [11]).

Below we prove that the Inclusive version of the protocol is at least as secure as the non-inclusive one, in terms of the probability of successful attacks. In addition, we show that the cost of an attack under Inclusive can be made high, by properly modifying the acceptance policy.

3.1 Acceptance Policy

The recipient of a given transaction observes the network's published blocks, and needs to decide when to consider the payment "accepted", that is, when it is safe to release the goods or services paid for by the transaction. He does so by making sure his transaction is included and confirmed by the main chain, and calculating the probability that it would be later excluded from it.

Probability of Successful Attacks We now compare the probability of a successful attack under the regular longest-chain protocol to the one under its Inclusive version. Our method applies to other main chain rules as well (e.g., GHOST). Recall that under Inclusive the blocks form a DAG, whereas when Inclusive is not implemented they form a tree (see Sect. 2). Notice that if $G(t)$ is the block DAG at time t, then if the network would have followed the non-inclusive setup, its block tree $T(t)$ would be precisely the subgraph of $G(t)$ obtained by removing all edges in blocks' reference list apart from the main edges (i.e., the first pointer in every block). For any DAG G let $F(G)$ be its main chain according to the underlying selection rule F (G can also be a tree).

Theorem 2. *Let $G(t)$ be the block DAG at time t, and let $T(t)$ be the block tree that is obtained from $G(t)$ by discarding the non-main edges. For any block $B \in F(G(t))$,*

$$\forall s > t : \Pr(B \notin F(G(s))) = \Pr(B \notin F(T(s))) \tag{2}$$

Proof. This is immediate from the fact that Inclusive does not change the way the main chain is selected, therefore, for all s: $F(G(s)) = F(T(s))$. □

As a corollary, the probability that a transaction would be excluded from the main chain does not become higher under Inclusive, as the security guarantees of main chain blocks apply to individual transactions as well (see the discussion succeeding Algorithm 1). In particular, any acceptance policy employed by a recipient of funds in a network following a non-inclusive protocol (see, e.g., [9, 11, 12]) can be safely carried out when Inclusive is implemented.

Cost of Attacks As mentioned at the beginning of this section, one may be interested in measuring the cost of a double-spend attack rather than its success probability. A potential drawback of including transactions from off-chain blocks is that it mitigates the cost of a failed double-spend attack. Double spend attacks consist typically of chains constructed by the attacker that are initially kept secret. The construction of blocks requires computational resources. Under the non-inclusive setup, when the attacker withdraws from the attack (usually after failing to build blocks faster than the network), its blocks are discarded. In contrast, under the Inclusive protocol, the attacker may still publish its secret chain and gain some value from transactions contained inside.

However, the recipient of funds can cancel this effect by waiting longer before accepting the payment. Indeed, if the attacker is forced to create long secret chains, its blocks suffer some loss due to the lower reward implied by the function $\gamma(\cdot)$.

To formalize this we provide first some definitions and notations. Denote by $G(t)$ the *published* developing block DAG at time t, and assume some main chain block B_{tx} confirms the transaction tx (that is, $tx \in \text{Inclusive-}F(G(t), B_{tx}, \emptyset)$). Let $H(t) \subseteq G(t)$ be the set of blocks from which B_{tx} is reachable, and denote the main chain atop B_{tx} (including itself) by $H_{main}(t) \subseteq H(t)$. Let $A(t) \subseteq G(t) \setminus H(t)$ be the set of blocks which satisfy $post(\cdot) \succ B_{tx}$; these are blocks which can be used by the attacker to reverse the transaction (even though the attacker did not necessarily create all of them), and the requirement on their $post(\cdot)$ block is to exclude from this set blocks earlier than B_{tx}, under the order of G (which do not affect the resolution of future conflicts).

Denote by val the expected reward from a block, under the Inclusive reward-scheme. val is equal, in equilibrium, to the expected cost of creating a block. We will simplify our analysis by assuming that val is constant. Finally, for convenience, we analyze the case where the underlying chain selection rule (F) is GHOST; the results apply to the longest-chain rule as well, after some slight changes.

Lemma 3. *Assume the attacker holds a fraction of at most q of the computational power. If $|H_{main}(t)| = n$, $|A(t)| = m$, and the attacker has created k secret blocks, then the cost of a failed attack satisfies*

$$cost \geq \sum_{h=m+1}^{m+k} (1 - \gamma(n + 2 - h)) \cdot val \tag{3}$$

Proof. In the best case for the attacker, its blocks form a chain which is built atop $A(t)$. If A_h is its hth block ($1 \leq h \leq k$) then $pre(A_h).height < B_{tx}.height - 1 + m + h$, or otherwise A_h necessarily references a block in H_{main} as its main parent (recall that a block's ordered reference list is forced to agree with F), and in particular it supports tx and does not participate in the attack.

In addition, the attacker's secret blocks are not published before the acceptance, hence their $post(\cdot)$ block height is at least $B_{tx}.height + n$. We conclude that the discount parameter on A_h is at most

$$\gamma\left((B_{tx}.height + n) - (B_{tx}.height - 1 + m + h - 1)\right),$$

hence its cost is at least $(1 - \gamma(n + 2 - m - h)) \cdot val$. After a change of parameter we arrive at (3). □

We now make use of this result to show that a payee that follows the acceptance policy introduced in [12] can make the attack cost arbitrarily high by waiting sufficiently before acceptance.

Corollary 4. *Let tx be a transaction in $G(t)$, and assume an attacker builds a secret chain that does not confirm tx, and that it persists with its attack as long as the payee has not approved the transaction. Then the minimal value of the* double-spend *needed for the attack to be profitable in expectation grows exponentially with t.*

Proof. Let $|H_{main}(t)| = n, |H(t)| = N$, and $|A(t)| = m$. The probability that an attacker with a fraction $q < 0.5$ of the computational power has managed to create k secret blocks is at most $e^{-q\lambda(t-t_0)}\frac{(q\lambda(t-t_0))^k}{k!}$, where t_0 is the time it began its attack. Following the dynamics of GHOST, the payee can wait for a collapse to occur, i.e., for B_{tx} to be included in the main chain of all honest nodes. Consequently, the probability that the attack will be successful is upper bounded by $\left(\frac{q}{1-q}\right)^{(N+1-m-k)^+}$. Here we used a worst-case assumption, according to which the attacker is able to exploit all of the blocks in $A(t)$ for its attack.

In case of a successful attack the attacker profits the amount double-spent, which we denote DS, while the profit from its blocks is offset by their creation costs. On the other hand, the cost of a failed attack is given by (3). Calculating the attack cost, we arrive at:

$$\begin{aligned}\text{attack-cost} \geq \sum_{k=0}^{\infty} e^{-q\lambda(t-t_0)}\frac{(q\lambda(t-t_0))^k}{k!} \cdot \Bigg(-DS \cdot \left(\frac{q}{1-q}\right)^{(N+1-m-k)^+} \\ + \left(1 - \left(\frac{q}{1-q}\right)^{(N+1-m-k)^+}\right) \cdot \sum_{h=m+1}^{m+k} (1 - \gamma(n+2-h)) \cdot val\Bigg)\end{aligned} \tag{4}$$

For a given time t, there is a probability distribution over DAGs that will be created by the network. This induces random variables for $N = N(t)$, $n = n(t)$,

and $m = m(t)$. As t grows these become arbitrarily close to their expected values (by the Law of Large Numbers). We can thus replace N with its expectation $(1-q)\lambda \cdot t$, and notice that $\mathbb{E}[n]$ grows with time and $\mathbb{E}[m]$ approaches a constant. Isolating DS shows that its minimal value in order for $\mathbb{E}$ [attack-cost] to be non-positive grows exponentially with t (assuming γ is non-trivial, that is, $\gamma \not\equiv 1$). □

To illustrate the growth of the attack cost, we show in Table 1 the minimal double-spend needed in order for an attack to be profitable in expectation. The table entries admit to the minimal DS making the attack profitable; here we fixed N and averaged over t (in contrast to the previous corollary). In addition, for simplicity we assumed $m = 0$ and $n = N$, corresponding to the case where the honest network suffers no delays. The penalty function γ_0 was selected as the one from Example 1, and the expected reward from a block were normalized so that $val = 1$. Notice that waiting for only one or two blocks is not safe at all, as the attacker can easily afford to try and create longer chains under the function γ_0 that we have chosen.

Table 1. The minimal double-spend (normalized by blocks' expected rewards, val) needed in order for an attack to be profitable in expectation, as a function of the number of confirmations and the attacker's computational power.

q	1	2	3	4	5	6	7	8	9	10
2%	0	0	$9.3 \cdot 10^2$	$1.2 \cdot 10^5$	$1.1 \cdot 10^7$	$8.3 \cdot 10^8$	$5.8 \cdot 10^{10}$	$3.8 \cdot 10^{12}$	$2.4 \cdot 10^{14}$	$1.3 \cdot 10^{16}$
6%	0	0	79	$3.1 \cdot 10^3$	$8.7 \cdot 10^4$	$2.1 \cdot 10^6$	$4.5 \cdot 10^7$	$9.1 \cdot 10^8$	$1.8 \cdot 10^{10}$	$2.9 \cdot 10^{11}$
10%	0	0	22	$4.8 \cdot 10^2$	$7.5 \cdot 10^3$	$9.9 \cdot 10^4$	$1.2 \cdot 10^6$	$1.4 \cdot 10^7$	$1.5 \cdot 10^8$	$1.4 \cdot 10^9$
14%	0	0	8.5	$1.3 \cdot 10^2$	$1.3 \cdot 10^3$	$1.2 \cdot 10^4$	$9.4 \cdot 10^4$	$7.1 \cdot 10^5$	$5.1 \cdot 10^6$	$3.2 \cdot 10^7$
18%	0	0	4.0	44	$3.3 \cdot 10^2$	$2.1 \cdot 10^3$	$1.2 \cdot 10^4$	$6.8 \cdot 10^4$	$3.6 \cdot 10^5$	$1.6 \cdot 10^6$
22%	0	0	2.0	18	$1.0 \cdot 10^2$	$5.1 \cdot 10^2$	$2.3 \cdot 10^3$	$9.7 \cdot 10^3$	$3.9 \cdot 10^4$	$1.4 \cdot 10^5$
26%	0	0	1.1	7.9	37	$1.5 \cdot 10^2$	$5.3 \cdot 10^2$	$1.8 \cdot 10^3$	$5.7 \cdot 10^3$	$1.6 \cdot 10^4$
30%	0	0	0.63	3.8	15	49	$1.4 \cdot 10^2$	$4.0 \cdot 10^2$	$1.0 \cdot 10^3$	$2.4 \cdot 10^3$
34%	0	0	0.36	1.9	6.4	18	45	$1.0 \cdot 10^2$	$2.3 \cdot 10^2$	$4.6 \cdot 10^2$
38%	0	0	0.20	0.92	2.8	6.9	15	30	58	$1.0 \cdot 10^2$
42%	0	0	0.10	0.43	1.2	2.6	5.2	9.3	16	25
46%	0	0	04	0.16	0.40	0.82	1.5	2.5	3.9	5.6
50%	0	0	0	0	0	0	0	0	0	0

The results above are not quite satisfying, as they demonstrate only the costs of an attack from a specific class: We assumed the attacker does not withdraw before the payee's acceptance. One could consider more sophisticated attack policies in which the attacker might withdraw earlier in order to reduce costs. The main obstacle here, is that there exist selfish mining strategies in which a miner profits from withholding some of his blocks, even under the non-inclusive setup ([7]). We point out that a malicious miner can execute double-spend attacks while employing selfish mining strategies, thereby guaranteeing itself an expected positive profit. While Inclusive protocols reduce the cost of a failed attack, we

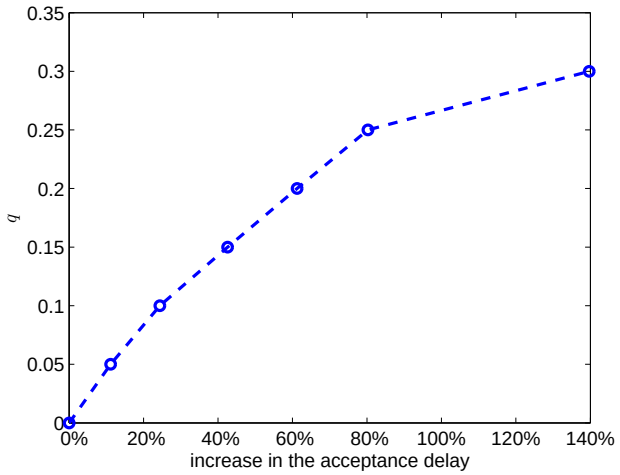

Fig. 1. The fraction of computational power an attacker needs to hold as a function of the increase in waiting time it aims to induce.

conjecture that adequate acceptance policies cancel this effect (as we have shown in Corollary 4 for one attack profile).

3.2 Delayed Service Attack

Another possible form of an attack is that of delayed service. The acceptance policy described above implies that if a recipient of a payment observes many blocks in the DAG that have the potential to form a competing main chain that will not accept his transaction, it must delay acceptance. Consequently, an attacker may decide to create its blocks deliberately off-chain, in attempt to increase the waiting time for transaction authorization in the network.

Notice that the attacker can never profit from a delayed service attack, say by reversing a previous payment, as its attack blocks are immediately published and are therefore transparent to any transaction authorizer. Moreover, the longer the attack goes on the greater its cost, as the gap between the $post(\cdot)$ and $pre(\cdot)$ of the participating blocks grows larger.

Assume the attacker wishes to delay the confirmation of transactions that lie in some block B. This can be done by increasing $|A(t)| = m$, that is, by publishing blocks from which B is not reachable. Despite the threat from $A(t)$, the honest network may add enough blocks to $H(t)$ for these transactions to be accepted.

We simulated this attack on a network with 100 equal miners, a delay of 2 seconds between each two, and a creation rate of 1 block per second. Figure 1 depicts the (fraction of) computational power needed by an attacker as a function of the increase in waiting time it aims to induce. The payees are assumed to use the policy induced by (4), with $q = 0.2$, and DS at most $1000 \cdot val$.

4 Transaction Selection under Inclusive Protocols

Up until now, we have not considered the effect of the Inclusive protocol on how participants choose the transactions they will include in their blocks. In fact, these choices are quite important: If all nodes choose the same subset of transactions for inclusion in their blocks, any two blocks that are created in parallel are likely to have many collisions, and throughput will not be high.

In this section we model transaction selection as a game, and show that nodes are actually incentivized to avoid collisions. They choose transactions with high fees, but will also compromise for lower fees with transactions that will have fewer collisions.

4.1 The Game Model

We model the process of embedding transactions in blocks as an infinite-horizon extensive form game, with N players (the miners), with imperfect information, i.e., players may be only partially aware of other players' moves (as they do not immediately see all the blocks that have been created; this is the main non-trivial aspect of the game). The game develops at discrete time steps $t = (1, 2, ...)$, with the gap between consecutive steps denoted Δ (where Δ is small).

We denote a transaction by w_i (or simply w) and ignore any property apart from its fee, which is assumed to fall into one of n discrete values, $v_1 > v_2 > \ldots > v_n > 0$ (fees in Bitcoin, for example, are specified in whole units of Satoshis). We write $v(w)$ to denote w's fee.

At every time step "nature" adds the same transactions simultaneously to all players' memory buffers (also known as *memory pools*). The number of new transactions is an independent random variable with mean $\eta\Delta$, for some $\eta > 0$. The fee of each new transaction is v_l with probability r_l, for some probability vector $\boldsymbol{r}$. If the size of the memory buffer of some player exceeds its limit $L > 0$, the transactions with lowest fees are dropped. Effectively, this means that nature's action space at every time step is finite, and can be mapped into $[n]^L$. Nature additionally chooses a (possibly empty) subset of players which will create a block at this time step. The probability that at a certain step player i will create a block is $\lambda_i\Delta$, with $\sum_{i=1}^{N} \lambda_i = \lambda$ being the network's block creation rate.

Player i observes only a partial signal of the actions of nature. He sees all new transactions,[10] and whether or not he was chosen to create a block. If so, he chooses a subset of his memory buffer of size at most b, where b is a positive integer constant representing the number of transactions per block. The chosen transactions are deleted from i's action space immediately, and from player j's action space after $t + d_{i,j}$ time steps, for some $N \times N$ integer matrix $(d_{i,j})_{i,j=1}^{N}$ (effectively deleting them from i and j's memory buffers). This simulates the delay in block propagation.

[10] This assumption approximates well the situation in the real Bitcoin network, in which transactions propagate quickly relative to blocks.

We are particularly interested in the case where the incoming rate of transactions exceeds the rate at which they are accepted into blocks (without this assumption, there is no scalability problem, and block sizes can be decreased).

A player may choose to use mixed strategies, namely, to select a distribution over the subsets of size b from his buffer. Instead of using distributions over a possibly exponential number of such subsets, it is more convenient to assign a probability (between 0 and 1) to every individual transaction in the buffer, such that the probabilities sum up to b. This scheme can be translated to probabilities over subsets (we show this in the full version of this paper). We adopt the latter approach, for its simplicity.

The Payoff Function Denote by $T(B)$ the set of transactions which block B was the first to contain, according to the order on blocks induced by Algorithm 1's run, denoted "$\prec$".[11] Then B's creator is awarded a fraction of $\sum_{w \in T(B)} v(w)$, as defined by $\gamma(c(B))$.

Finally, as is usually customary in infinite horizon games, a discount factor $0 < \beta < 1$ is applied to all rewards, such that if a player has created blocks $B_1, B_2, ...$ at time steps $t_1, t_2, ...$, his reward from the game is $\sum_j \beta^{t_j} \cdot \gamma\left(c(B_j)\right) \cdot \sum_{w \in T(B_j)} v(w)$.

4.2 Rationality in the Inclusive-*F* Game

The solution concept that best matches our scenario (in which players have partial information about the recent actions of others) is the *sequential equilibrium* which was developed by Kreps and Wilson [8]. This concept explicitly considers the beliefs of players about the history and current state of the game. Intuitively, the sequential equilibrium concept ensures that a single player does not expect to benefit from deviating (given these beliefs). Threats are additionally "credible" and behaviors are temporally consistent (this is similar to sub-game perfection). Finally, players' beliefs about the state of the game are required to be "consistent".

We extend the result in [5] to the infinite horizon setting and show the existence, for all $\epsilon > 0$, of an ϵ-perfect sequential equilibrium in our game (in which players who deviate may gain, but no more than ϵ).

Lemma 5. *For every $\epsilon > 0$ there is an ϵ-perfect sequential equilibrium in the Inclusive-F game.*

[11] To make this formal some work is needed: Let $G(t)$ be the block DAG which consist of all blocks created up to time t. We require that the underlying chain selection rule F break ties between equally weighted leaves, in some predetermined perhaps arbitrary way. Denote by B_t the leaf of the main chain $F(G(t))$. Assume F converges, in the sense that a block in the main chain becomes less likely to be replaced, as time grows: $B \in F(G(t)) \implies \lim_{s \to \infty} \Pr(B \notin F(G(s))) = 0$ (longest-chain and GHOST, for instance, satisfy this property). We can thus speak of the eventual- or limit-order "$\prec$" on all blocks in the history of the game, defined by $A \prec A'$ if $\exists t_0, \forall t > t_0 : A \prec_{B_t} A'$ (see the discussion succeeding Algorithm 1 for the definition of $\prec_{B_t}$).

We prove this in the full version of this paper.

Note that several equilibria may (and do) exist, and worse yet, while the proof of existence is constructive, it requires the exploration of an exponentially large state space (essentially enumerating all possible subsets of transactions that will enter the buffer in the future). We therefore desire an efficient algorithm that will preform well in practice.

4.3 Myopic Strategies

We restrict the discussion in this subsection to a simplified version of the game, namely, the single shot game. In this setup, when a player chooses transactions for his current block, he disregards the effect this choice may have on which transactions will be available for his next block. In addition, we assume all players have identical buffers of transactions to choose from. Finally, we assume that a block's position within the block DAG does not depend on its creator's identity.

This simplified model can be seen as a good approximation to an adequately distributed network, in which individual players hold a small fraction of the total computational power. A small player does not create blocks often, and thus his current block has very little effect on his future rewards.

A Myopic Equilibrium For any block B let $pconf(B)$ denote the set of blocks which precede B in the order "$\prec$" but are not reachable from it. Assume that all players include transaction w in their block (if the block is indeed created) with a marginal probability p_w; then B's expected reward from selecting w is $w \cdot (1-p_w)^{|pconf(B)|} \cdot \mathbb{E}[\gamma(c(B)) \mid |pconf(B)|]$. We define,

$$f(p_w) := \sum_{l=0}^{\infty} \Pr(|pconf(B)| = l) \cdot (1-p_w)^l \cdot \mathbb{E}[\gamma(c(B)) \mid l].$$

One could verify that $w \cdot f(p_w)$ is the player's expected reward from embedding w in B. Note that f is strictly decreasing in p_w, and so its inverse f^{-1} exists.

Theorem 6. *Suppose the memory buffer consists of k_l transactions with fee v_l $(1 \le l \le n)$. Denote the individual transactions by $w_1, \ldots, w_m$, which are sorted in descending order of their fees. Denote the index of $v(w_i)$ by $l(w_i)$. The marginal probability $p_i := \frac{q_{l(w_i)}}{k_{l(w_i)}}$ $(1 \le i \le m)$ defines a symmetric equilibrium in the single-shot inclusive-F game, where:*

- $q_l = \begin{cases} k_l \cdot \min\left(f^{-1}\left(\frac{c_{k_{max}}}{v_l}\right), 1\right) & 1 \le l \le k_{max} \\ 0 & k_{max} < l \le n \end{cases}$
- $\forall\, 1 \le l \le n$: $G_l(z) := \sum_{h=1}^{l} k_h \cdot \min\left(f^{-1}\left(\frac{z}{v_h}\right), 1\right) - b$
- $k_{max} := \max\{k \le n \mid \forall l \le k : G_l(v_l) \le 0\}$
- $c_{k_{max}}$ *is the root of* $G_{k_{max}}$.[12]

[12] Note that $k_{max} \ge b$, and that the existence of a root for $G_{k_{max}}$ follows from the fact that f's domain is $[0,1]$ hence this is also f^{-1}'s image.

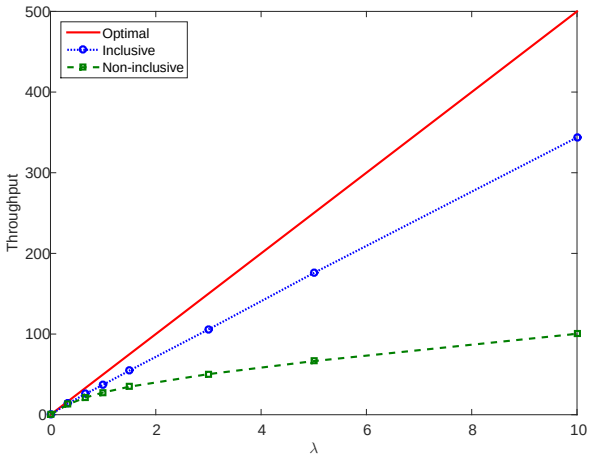

Fig. 2. The fraction of optimal throughput achieved in Inclusive and non-inclusive longest-chain protocols.

The proof is deferred to the full version of this paper. In Sect. 5 we show that this strategy performs well, in terms of throughput and utility, despite the simplifying assumptions used to derive it.

In addition to the analysis above, we also explored other solution concepts, namely, safety-level strategies and cooperative strategies that maximize the social welfare. They are discussed in Appendix A. In all cases, players use randomized strategies to select transactions for their blocks in a way that results in an increase in throughput.

5 Implications of Inclusive Protocols

5.1 Throughput

The throughput of the system, when the Inclusive protocol is implemented, depends on the behavior of the players. We demonstrate Inclusive's ability to achieve significantly higher results by checking the throughput when the players act according to the myopic strategy defined above.

We simulated a network with 100 identical players. The distance between each pair of players was a constant $d = 1$ second. We examined different block creation rates λ varying from 0 to 10 blocks per second. Block sizes were set to $b = 50$ transactions per block. The transaction arrival rate was 65 transactions per second, and their fees drawn uniformly from [0,1]. In each simulation we compared the performance of the myopic strategy to the non-inclusive outcome. We compare the resulting throughput to the optimal achievable rate, which is achieved in centralized networks with no delays. Figure 2 depicts the results, showing substantial gains over the non-inclusive protocol.

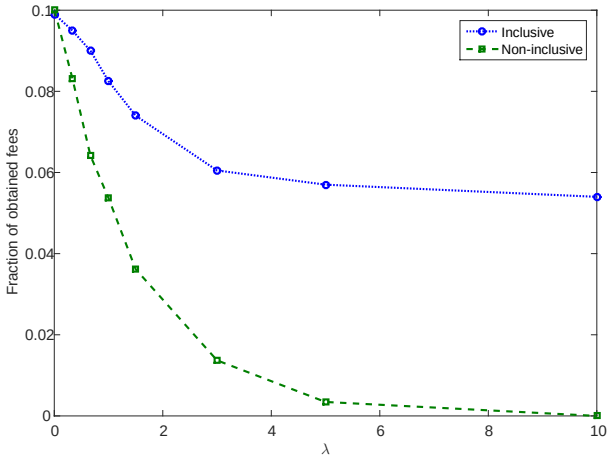

Fig. 3. The fraction of rewards obtained by a weak (10%) miner under delays.

5.2 Fairness

While a miner with computational power $q\lambda$ owns a fraction q of blocks in the block DAG (in expectation), highly connected miners will have more of their blocks in the main chain compared to poorly connected ones. This phenomenon lowers the profitability of weak players that are unable to match the return on investment enjoyed by larger ones, and slowly pushes Bitcoin towards an increased concentration of mining power. Given two miners with equal connectivity, but differing hash rates, the larger miner of the two also enjoys an advantage as he immediately begins to extend his own block using more computational power than his weaker opponent.

Inclusive protocols significantly mitigate this effect. Off-chain blocks reward their owners with some fees, so weak or poorly connected miners, who have a higher proportion of off-chain blocks, suffer fewer losses.

Consider, for instance, a network with two strong miners each owning a fraction 0.45 of the total computational power, and a weak miner owning 0.1. We simulated this scenario, and examined the revenue of the small miner. The results are given as a fraction of the social welfare, in Fig. 3, and show a significant mitigation of the nonlinearity phenomenon.

6 Related Work

The Bitcoin protocol was published by Satoshi Nakamoto in a white paper in 2008 [9]. The security analysis in the paper was later improved in [11]. The propagation of large blocks in the network was first studied in [6], where empirical measurements and analysis have shown that larger blocks conflict more often, and some economic implications such as the desire of miners to create smaller blocks was considered. Additional analysis of phenomena related to larger block sizes was given in [12]. The incentives of miners to propagate transactions was

studied in [2]. A recent work by Eyal and Sirer has shown that large miners may choose not to follow the exact protocol and may delay the propagation of their own blocks in order to increase their revenue [7]. These effects still persist in our own version of the protocol, and so we assume that honest nodes follow the protocol without such manipulations.[13]

Additional techniques to mitigate the effects of an increased number of transactions on the network include the proposal for micro-transactions channels (see, e.g., [4]). These channels effectively allow two transacting parties to open a micro-payment channel by freezing some sum of money and transmitting it in small quantities, effectively updating a transaction that includes the total transfer thus far. The aggregating transaction is committed to the block chain after some time has passed. Micro-transaction channels are not as useful in second generation protocols, as they are not suitable to updates that cannot be easily aggregated. In addition, the costs of locking money in advance and the limitation to channels between pairs of nodes further restrict the use of this approach. Other discussions in the Bitcoin community include the use of invertible Bloom filters to reduce the amount of information transmitted between nodes [1].

As our work considers structural changes to the block chain structure, it is also worthwhile to mention proposals such as Side Chains [3] that are currently being discussed in the Bitcoin community.

7 Conclusion

We presented the Inclusive protocol that integrates the contents of off-chain blocks into the ledger. Our modification results in incentives for behavior changes by the nodes that lead to an increased throughput, and a better payoff for weak miners. Our plans for future work include additional analysis of transaction authorization policies and waiting times as well as evaluations of the protocol under selfish mining.

8 Acknowledgements

The authors were supported in part by the Israel Science Foundation (Grants 616/13, 1773/13 and 1227/12), by the Israel Ministry of Science and Technology (Grant 3-6797), and by the Israel Smart Grid (ISG) Consortium.

References

1. Andresen, G.: O(1) block propagation, https://gist.github.com/gavinandresen/e20c3b5a1d4b97f79ac2
2. Babaioff, M., Dobzinski, S., Oren, S., Zohar, A.: On Bitcoin and red balloons. In: The 13th ACM Conference on Electronic Commerce. pp. 56–73. ACM (2012)

[13] Successful manipulations require strong attackers that are either highly connected, or have massive amounts of computational power.

3. Back, A., Corallo, M., Dashjr, L., Friedenbach, M., Maxwell, G., Miller, A., Poelstra, A., Timón, J., Wuille, P.: Enabling blockchain innovations with pegged sidechains (2014)
4. bitcoinj: Working with micropayment channels, https://bitcoinj.github.io/working-with-micropayments
5. Chakrabarti, S., Topolyan, I.: A direct proof of the existence of sequential equilibrium and a backward induction characterization (2010)
6. Decker, C., Wattenhofer, R.: Information propagation in the Bitcoin network. In: 13th IEEE International Conference on Peer-to-Peer Computing (P2P), Trento, Italy (September 2013)
7. Eyal, I., Sirer, E.G.: Majority is not enough: Bitcoin mining is vulnerable. In: Financial Cryptography and Data Security, pp. 436–454. Springer (2014)
8. Kreps, D.M., Wilson, R.: Sequential equilibria. Econometrica: Journal of the Econometric Society pp. 863–894 (1982)
9. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
10. Nisan, N., Roughgarden, T., Tardos, E., Vazirani, V.V.: Algorithmic game theory, chap. 9. Cambridge University Press (2007)
11. Rosenfeld, M.: Analysis of hashrate-based double spending. arXiv preprint arXiv:1402.2009 (2014)
12. Sompolinsky, Y., Zohar, A.: Secure high-rate transaction processing in Bitcoin. In: Financial Cryptography and Data Security. Springer (2015)
13. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger (2014)

A Additional Game Theoretic Analysis

A.1 Safety Level

As the players' behavior is unknown and can take different courses, one may be interested in the player's *safety level*, namely, the minimal utility he can guarantee himself. In the worst case for the player, the rest of the players choose a strategy which minimizes his utility, and the safety level is his best response to such a scenario.

Formally, player i's safety level is the solution to the zero-sum game, where i is the max-player while the rest of the network acts as his united adversary min-player. The following theorem provides the player with a marginal probability over his memory buffer, which serves as his maxmin strategy for the single-shot game at time t.

Theorem 7. *Denote player i's memory buffer by $w_1, \ldots, w_m$ (sorted in descending order of their fees) at a time in which it was able to create a block. Denote $\delta := 2 \cdot \max_j\{d_{i,j}\} \cdot (\lambda - \lambda_i)$, and for all $q \in [0,1]^m$ define $f(q) := \sum_{k=1}^{m} q_k \cdot \left(w_k e^{-\delta} \sum_{l=0}^{\lceil \frac{k}{b} \rceil - 1} \frac{\delta^l}{l!} \right)$. Let q^* be the solution of the next linear program:* $\max f(q)$ *s.t.* $\forall k < m: \; q_k w_k \geq q_{k+1} w_{k+1} \; ; \; \forall k: \; 0 \leq q_k \leq 1 \; ; \; \sum_{k=1}^{m} q_k = b$. *Then i's utility from q^* is at least $f(q^*)$, regardless of the strategy profile of the other players.*

The idea behind the proof is to construct a game in which player i chooses transactions for his blocks, while the rest attempt to choose the very same transactions. In the worst case scenario for the player, his rivals correlate their blocks' contents so as to maximize collisions with i's blocks. Another worst case assumption is that the delay between the players and i is maximal. Refer to the full version of the paper for a formal construction and proof of the theorem.

A.2 An Optimal Strategy

The performance of any solution of the game, including those considered thus far, should be compared to the optimal setup. If players would play cooperatively, so as to try and maximize the system's performance, then all blocks would contain unique transactions, with the top most fees available. Formally, if n blocks were created by the network during some long time period T, then the system's hypothetical optimal performance, $OPT(T)$, is defined as the sum of the top $n \cdot b$ transactions created within T (this is not necessarily feasible, as high transactions may not be available to early blocks).

Recall that transactions arrive at a rate of η. Their values are drawn according to some probability vector $\boldsymbol{r}$, and we denote by R the corresponding CDF. The rate at which transactions are embedded in the DAG is denoted $\lambda_{out} := b \cdot \lambda$ (it is the hypothetical optimal throughput).

We define a threshold below which transactions are totally ignored by the players: $v_{thresh} = R^{-1}(1 - \frac{\lambda_{out}}{\eta})$. This threshold defines a cutoff, $\theta := \{j : v_j > v_{thresh}\}$. We claim that if players choose transactions above this cutoff, *uniformly*, then the resulting social welfare, which is the throughput weighed according to fees, would coincide with $OPT(T)$, as T goes to infinity. We denote the described strategy by UCO (Uniform above CutOff), and by $UCO(T)$ the total weighed throughput achieved by applying UCO up to T.

Proposition 8. *Assume nodes have an unlimited memory buffer. Let T be some time window, and denote by $n(T)$ the number of blocks that were created by that time. Then, $\lim_{T\to\infty} \frac{1}{n(T)} \cdot \mathbb{E}[OPT(T)] = \lim_{T\to\infty} \frac{1}{n(T)} \cdot \mathbb{E}[UCO(T)]$, where the expectation is taken over all random events in the network and in the realization of UCO.*

The intuition behind the result is that choosing a cutoff as we have prescribed implies that the incoming and outgoing rates of transactions to the buffer are equal. Thus, results from queueing theory show that the expected size of the buffer is infinite, and miners always have enough transactions above the cutoff to include in blocks. In particular, for large enough memory buffers, there are effectively no collisions between different blocks, and the transactions in blocks are unique. This surprising result is achieved at a cost: transactions have long expected waiting times for their authorization.