Sliding Window Challenge Process for Congestion Detection

Ayelet Lotem[1], Sarah Azouvi[2], Patrick McCorry[3] and Aviv Zohar[1]

[1]The Hebrew University of Jerusalem, {ayelem02,avivz}@cs.huji.ac.il
[2]Protocol Labs, sarah.azouvi@protocol.ai
[3]Infura, stonecoldpat@gmail.com

Abstract. Many prominent smart-contract applications such as payment channels, auctions, and voting systems often involve a mechanism in which some party must respond to a challenge or appeal some action within a fixed time limit. This pattern of challenge-response mechanisms poses great risks if during periods of high transaction volume, the network becomes congested. In this case fee market competition can prevent the inclusion of the response in blocks, causing great harm. As a result, responders are allowed long periods to submit their response and overpay in fees. To overcome these problems and improve challenge-response protocols, we suggest a secure mechanism that detects congestion in blocks and adjusts the deadline of the response accordingly. The responder is thus guaranteed a deadline extension should congestion arise. We lay theoretical foundations for congestion signals in blockchains and then proceed to analyze and discuss possible attacks on the mechanism and evaluate its robustness. Our results show that in Ethereum, using short response deadlines as low as 3 hours, the protocol has $> 99\%$ defense rate from attacks even by miners with up to 33% of the computational power. Using shorter deadlines such as one hour is also possible with a similar defense rate for attackers with up to 27% of the power.

Keywords: Congestion, Challenge-Response

1 Introduction

Defi platforms constructed over blockchains such as Ethereum have seen a recent boom of activity and interest. Their growing ecosystem allows for increasingly complex financial interactions executed in a fully decentralized manner. The main building block used to construct these platforms are the smart contracts that define the rules of interaction in code.

Smart contracts enable a wide range of applications, such as auctions, voting systems, and second layer protocols (e.g., payment channels) that operate above the blockchain layer. They typically provide rules that allow them to act as an automated adjudicator in case conflicts between participants arise.

For many applications, interactions with smart contracts are time dependent and are even subject to deadlines, meaning that in some cases, transactions

added after a specific moment will effectively be rejected. For example in the case of auctions, a bid must be received before the end of the auction otherwise it is not valid. Another example appears in the context of payment channels [12] where participants have a limited interval of time to dispute the division of funds if they disagree with their peers.

A major weakness of such deadlines is that in cases in which the blockchain is congested, users that submit transactions will not have them included in blocks in time. In fact, several attacks and failures can be attributed directly to this weakness (we provide some examples below). One mitigation often employed by participants is to offer higher fees for transactions with deadlines which means users are usually overpaying. Another is to extend the deadlines which greatly delays processing and settlement within the context of the relevant smart contract. In many cases, transaction fees and deadlines are decided upon in advance, before knowing the exact conditions that will prevail when the transaction is actually transmitted, which causes participants to take wider safety margins and increases costs further. Due to the well-known scalability issues of blockchains [1], we expect congested periods to become increasingly more common, which will directly impact the design of time-sensitive smart contracts.

Our contributions: In this work we present a mechanism aimed at solving these issues. We propose to set short deadlines that are automatically extended if congestion occurs. We lay the theoretical foundations of congestion monitoring in blockchains and formalize the notion of challenge-response protocols in this context. We then propose two different protocols to detect congestion over multiple blocks: The 'L Consecutive Blocks' protocol defines uncongestion by the existence of L consecutive uncongested blocks, and its generalization, the 'Sliding Window (K-out-of-N)' protocol which defines uncongestion by the existence of N consecutive uncongested blocks with K uncongested blocks among them. We show that the Sliding Window protocol is more resilient to attacks than the L-Consecutive Blocks protocol when attacked by miners. We propose a new opcode for Ethereum that will provide the required functionality, and also provide an implementation (not requiring new opcodes) in Solidity, using opcodes introduced by EIP 1559 [5].

Examples of congestion attacks and related failures. A recent well known example of congestion related failure took place on *Crypto Black Thursday* (March 12th, 2020) when the price of Ethereum dropped by more than 50% in less than 24 hours [11]. This led to a panic-sale of coins and hence increased congestion. At the peak, during a 2-3 hour window the Ethereum blockchain's fees climbed to $1.65 on average, which is more than 10 times more than in the previous days.

The drop in ETH price triggered many MakerDAO auctions to liquidate collateral (typically collateral on short positions must be sold if prices fluctuate too much). The tokens to be sold were purchased at almost no costs due to the inability of many bidders to send transactions and participate. This has, allegedly, been used by one user to gain $8.3 million worth of Ether [3].

Several studies [10,13] deal with different types of attacks designed to prevent a party from responding on time to a challenge [10]. Harris and Zohar [13] present

an attack where the attacker forces many victims at once to flood the blockchain with claims for their funds. The ensuing congestion allows the attacker to steal the funds that cannot be claimed before the deadline. Our protocol will prevent these issues by extending the deadlines until the congestion passes.

2 Related Work

Congestion is a real-world problem faced by the most prominent cryptocurrencies. In addition to the popular examples of Crypto Black Thursday or Cryptokitties, widely discussed online [3,11,6], Sokolov [17] examined periods of congestion caused by ransomware.

One way to deal with congestion, is to improve the scalability of the underlying consensus protocol [20,19,18,7,9] or to introduce higher level layers that help to scale. Solutions ranging from sharding [22], off-chain payment channels [12] or layer zero optimization [21] (i.e., network level optimization) have been considered. All these solutions improve the number of transactions per second that the network can process, but congestion may still occur even at higher rates.

Other methods that help to ensure that time-sensitive transactions are processed are rather ad-hoc. For example, the *replace by fee* mechanism [4] and *child pays for parent* [2], allow users to add or change the fees of their transactions. Bitcoin's fee mechanism - equivalent to a first price auction - is sub-optimal and often results in users paying more than what is necessary. Ethereum Improvement Proposal 1559 (EIP 1559) was made to change this mechanism in Ethereum [16,5]. EIP 1559 implements a *base fee* which is burned. This base fee can be seen as an indication of the level of congestion in recent blocks, and we utilize this in our implementation.

Another line of research that could potentially prevent transaction fees from spiking considers order-fairness consensus protocols [14,15]. The idea is to ensure that transactions are ordered in the blockchain in the same order they arrived in. This also helps avoiding problems such as front-running [8].

3 Preliminaries and Definitions

3.1 Challenge-Response Protocols

A challenge-response protocol is an implementation of a pattern in which some party must respond to a challenge within a fixed time limit. This pattern consists of a challenge that takes effect at time T_c and a response deadline T_{rd} which is the latest time by which response to the challenge will be accepted. We call the time period between T_c and T_{rd} the challenge window. Responding to the challenge during the challenge window yields different results compared to responding *after* the deadline. The protocol we propose inspects the challenge window period and extends it (by extending T_{rd}) as long as the blockchain stays congested.

3.2 Blockchain Congestion

Our protocol has two components. First it relies on a mechanism to define what it means for a block to be congested. We then use this definition to define an uncongested *period*. Intuitively, a period will be (un)congested if some threshold of blocks is (un)congested. We start by defining block congestion before moving on to presenting different period congestion definitions and choosing one that meets our requirements.

Blocks and Transactions. A block $\mathbf{B} = \{\mathsf{tx}_1, \cdots, \mathsf{tx}_n\}$ is as a set of transactions (we ignore the order of transactions in the block as well as other data - such as nonce - as they are irrelevant to our problem). Transactions pending to be included in a block are kept locally by each node in their *mempool* until they are included in the chain. Each transaction tx has a size $w(tx)$, and a fee density $\phi(tx)$. The fee paid by the transaction is therefore $w(tx) \cdot \phi(tx)$. We define the total weight of transactions in a block $\mathbf{B}$ with a fee density above θ as $\mathcal{W}_{\mathbf{B}}(\theta) \coloneqq \sum_{tx \in B:\ \phi(tx) \geq \theta} w(tx)$.

Blocks can contain transactions with total size bounded by $\mathcal{B}$, i.e., $\mathcal{W}_{\mathbf{B}}(0) \leq \mathcal{B}$. For simplicity we treat every block as *full*, i.e., for any *block* $\mathcal{W}_{\mathbf{B}}(0) = \mathcal{B}$ (if necessary we fill them artificially with transactions with a fee of 0).

The total amount of fees collected from a block by the miner is $\mathcal{U}_{\mathbf{B}} \coloneqq \sum_{tx \in B} w(tx) \cdot \phi(tx)$. If the size of the mempool is bigger than the maximum block size $\mathcal{W}_{\mathbf{B}}(0)$, we assume that honest miners choose the transactions in a way to maximize the fees they get.

Period. A period $Pe = (b_1, b_2, ..., b_n)$ in the blockchain is a non-empty sequence of **consecutive** blocks. We denote the length (number of blocks) of the period by $|Pe| = n$, and write for $i \in \{1, ..., n\}$: $Pe[i] = b_i \in Pe$. For a period P_2, we say that period P_1 is included in P_2 and note $P_1 \subseteq P_2$ if every block in P_1 is included in P_2.

In this work, we want to capture the notion of congestion: a phenomenon where there's a spike in the number of transactions waiting in the mempool. Since the mempool is not part of the blockchain, we instead rely on the data in the blocks in order to define congestion. We propose the following definition for *block congestion*.

Definition 1 ((θ, γ)-congestion). *We say that a single block $\mathbf{B}$ is (θ, γ)-congested if $\mathcal{W}_{\mathbf{B}}(\theta) \geq \gamma \cdot \mathcal{B}$ and denote $\mathcal{C}_{\theta,\gamma}(\mathbf{B}) = 1$, where $\mathcal{C}_{\theta,\gamma}$ is the corresponding indicator function.*

The definition above examines all transactions above fee density θ and require that they make up at least a γ-fraction of the block in terms of size. Intuitively, for $\gamma = 1$ the definition captures that if a block is $(\theta, 1)$-congested, a transaction needs to have a fee density that is at least θ in order to have a better chance of being included. In other words, we use the price of entering a transaction to the blockchain as a reliable signal for congestion.

For a block $\mathbf{B}$ and a fee density $\theta \geq 0$, we define the *θ-weight threshold* $\gamma_{\mathbf{B}}(\theta)$, as the maximum fraction of the block weight under which the block is (θ, γ)-congested. From definition 1 it is clear that $\gamma_{\mathbf{B}}(\theta) = \frac{\mathcal{W}_{\mathbf{B}}(\theta)}{\mathcal{B}}$. Similarly, for a block $\mathbf{B}$ and a fraction $\gamma \geq 0$, we define the *γ-fee density threshold* $\theta_{\mathbf{B}}(\gamma)$, as the maximum fee density under which the block is (θ, γ)-congested ($\theta_{\mathbf{B}}(\gamma) \coloneqq \max\{\theta \mid \mathcal{C}_{\theta,\gamma}(\mathbf{B}) = 1\}$).

Block manipulation. One of the key measures we are interested in is when is an adversary able to manipulate blocks' congestion signals. When a miner mines a block, they can choose to include transactions from their mempool, or add dummy transactions that move money between their accounts and pay a fee (to themselves), making the fees appear different than they ought to be. However, miners cannot manipulate blocks at arbitrary heights, and doing so would incur a cost. The miner's chance of mining a new block depends on its relative computational power. Therefore, as is standard, we denote the computational power of an adversary by α. Each block has a probability α to be mined by the adversary, and $1 - \alpha$ to be mined by the other miners. Furthermore, giving up mempool transactions means missing out their fees and hence induces a loss that we compute in the next two propositions.

Proposition 1. *An adversary manipulating a block $\mathbf{B}$ to make it (θ_1, γ_1)-congested when it is not, will lose a potential profit of at-least $\mathcal{B} \cdot \int_{1-(\gamma_1 - \gamma_{\mathbf{B}}(\theta_1))}^{1} \theta_{\mathbf{B}}(\gamma)\, d\gamma$.*

Proposition 2. *An adversary manipulating a block $\mathbf{B}$ to reverse its signal from (θ_1, γ_1)-congested to not will lose a potential profit of at-least $\mathcal{B} \cdot \int_{\gamma_1}^{\gamma_{\mathbf{B}}(\theta_1)} (\theta_{\mathbf{B}}(\gamma) - \theta_1)\, d\gamma$.*

The proofs for both propositions can be found in Appendix A.1.

Before moving on to define period congestion, we note that there exist other ways in which block congestion could be defined. For example, in Section 5, we use EIP 1559 *base fee* as a measure of congestion and use it to implement our suggested protocol. We include several other examples that are less efficient in Appendix B.

Congestion vector of a period. To determine whether a period Pe is uncongested we will refer to the congestion vector $Pe^c \coloneqq (\mathcal{C}(Pe[i]))_{i=1}^{n} \in \{0,1\}^n$ which consists of the congestion signal of its blocks. Intuitively, if most of the blocks in the period are congested then the period is congested and vice-versa. However, we must also account for the fact that an adversary may be able to change the congestion signal of some of the blocks, as already discussed. We will consider different protocols to define period uncongestion, a situation in which the period is considered not congested. An uncongestion period protocol is a function that we denote: $\mathsf{UCP}{:}\{0,1\}^* \to \{0,1\}$. This function takes as input a binary series representing the congestion signal of the blocks in the examined time period. It will return 0 if the period is congested and 1 otherwise. This function can furthermore be extended to also provide auxiliary information such

as a proof π in the case where the period is uncongested. For the efficiency of the protocol, we will strive for a definition that can provide a compact and easy-to-verify proof. In the rest of the paper, we use $B(n,p)$ to denote the Binomial distribution with parameters n and p.

Definition 2 (Period Manipulation). *For a period Pe and an adversary with a fraction α of the total computational power, we associate a manipulated period $\hat{Pe}$ defined as follows. For $i \in \{1,\ldots,|Pe|\}$ the adversary can replace $Pe[i]$ with probability α, with a block that has a congestion signal of their choice. We denote by $\overline{m} = m_{|Pe|}(\alpha) \sim B(|Pe|,\alpha)$ the vector which indicates which of the blocks in the given period the adversary controls, meaning the adversary can replace the $Pe[i]$ block's congestion signal iff $\overline{m}[i] = 1$. We then define the adversary's manipulation set $S_{\overline{m},Pe} \coloneqq \{\hat{Pe^c} \in \{0,1\}^{|Pe|} \mid \forall 1 \leq i \leq |Pe| : \overline{m}[i] = 0 \Rightarrow \hat{Pe^c}[i] = Pe^c[i]\}$. Intuitively, $S_{\overline{m},Pe}$ corresponds to the set of possible congestion vectors that the adversary could create by changing the signal of the blocks that it controls.*

In a real world setting, even if there is a long period of uncongestion, it could be the case that one or more of the blocks are fuller than the others due to some randomness in the transactions' arrival time (e.g., there was a temporary high transaction volume). To account for this randomness, we make a simplifying assumption that blocks are congested independently with probability p and say that the blockchain is *$p-$congested*. We note that in reality, congestion is often changing, and is usually correlated when considering several consecutive blocks. We leave more complex models of congestion for future work. In our case, the congestion vector of a period Pe chosen at random has a binomial distribution: $Pe^c \sim B(n,p)$. When studying attacks where the adversary tries to convert a congested period to an uncongested one, we will assume that p is close to one (i.e. most of the blocks are congested), whereas when studying the opposite case, we will consider p to be close to zero.

Our protocol consists in extending the deadline of challenge-response in the event of a congestion period. However, to avoid an edge case where the deadline is extended indefinitely, we define $\hat{M}$ - an upper bound on the total length of the extended period.

Definition 3 ($\hat{M}$-Maximum Extension). *Given a challenge-response protocol in a $p-$congested blockchain where the challenge starts at block height h, we say that $\hat{M}$ is the Maximum Extension of the challenge if the deadline cannot be extended further than height $h + \hat{M}$.*

3.3 Desirable Properties of Protocols

We define some properties that we aim for our protocol to achieve.

In the rest of the paper we use the following notation: $D \leftarrow s$ to denote that s was selected randomly from the distribution D. We start by defining the two attacks that we will consider: congestion attack and uncongestion attack before defining the *robustness* of the protocol, that captures the security of the protocol against either attack.

Definition 4 (Congestion/Uncongestion attack on Pe). *Given a period Pe, chosen at random in a p-congested blockchain, we say that the adversary wins a congestion, resp. uncongestion, attack on Pe if it can manipulate Pe into an uncongested, resp. congested, period.*

Definition 5 ((α, p, q, n)-congestion robustness). *We call an uncongestion period protocol UCP:$\{0,1\}^* \to \{0,1\}$, (α, p, q, n)-**congestion robust** if given an adversary with a relative computational power α, his probability of winning a congestion attack, i.e. successfully manipulating a period Pe of n blocks into a congested period $\hat{Pe}$, is less than q.*

$$B(n,p) \leftarrow Pe \ : \ P_r(\exists \hat{Pe} \in S_{\overline{m},Pe} \ s.t. \ \mathsf{UCP}(\hat{Pe}) = 0) \leq q.$$

Definition 6 ((α, p, q, n)-uncongestion robustness). *We call an uncongestion period UCP:$\{0,1\}^* \to \{0,1\}$, (α, p, q, n)-**uncongestion robust** if given an adversary with a relative computational power α, his probability of winning an uncongestion attack, i.e. successfully manipulating a period Pe of n blocks into an uncongested period $\hat{Pe}$, is less than q.*

$$B(n,p) \leftarrow Pe \ : \ P_r(\exists \hat{Pe} \in S_{\overline{m},Pe} \ s.t. \ \mathsf{UCP}(\hat{Pe}) = 1) \leq q.$$

Definition 7 (Monotonicity). *A congestion protocol is **monotone** if for every two periods Pe_1 and Pe_2, if $Pe_1 \subseteq Pe_2$ and Pe_1 is considered uncongested, then so is Pe_2, i.e., $\forall Pe_1 \subseteq Pe_2 : \mathsf{UCP}(Pe_1^c) = 1 \to \mathsf{UCP}(Pe_2^c) = 1$.*

Intuitively, a monotone protocol is easier to verify as the prover only needs to select a portion of blocks from the time period Pe in order to prove uncongestion. Furthermore, a monotone protocol requires only sporadic access to the blockchain. A prover can go offline and prove uncongestion when they come back online by choosing any uncongested period from the time they were offline. In the case of a non-monotonic protocol, if the prover is offline during an uncongested period, they cannot prove the uncongestion of the longer period, after they came back online, they missed the uncongested period.

Efficiency Properties. We define two properties that capture the efficiency of the protocol.

- **Concise proof size** The evidence needed to prove uncongestion of a period should be as concise as possible.
- **Concise refresh information** The extra information needed to be kept when checking the congestion signal of a period that has already been extended due to congestion should be as concise as possible. Intuitively, when we extend a period, from Pe_1 to Pe_2, in order to check Pe_2 for congestion, we should not have to re-check every block in Pe_1 but, ideally, aggregate this information.

In the next section we will discuss different period congestion protocols with the goal of finding one that will be proof efficient and robust against an attacker with reasonable hash rate with high probability.

4 Uncongested Period Protocols

We examine different protocols that fit the definition of congestion of a period Pe. We start by presenting "naive" protocols and see why they are not good enough, i.e., why they lack the desirable properties defined in Section 3.3.

4.1 Strawman Protocols

Definition 8 (Cumulative M). *Period Pe is uncongested if there exists M blocks which are uncongested:* $\mathsf{UCP}_{CM}(Pe^c) = 1 \leftrightarrow \left(\sum_{b \in Pe}(1 - \mathcal{C}(b)) \geq M\right)$.

This protocol is monotonic but is not sufficiently robust to adversarial attacks: if we wait long enough, the probability of the adversary controlling M blocks becomes overwhelming (even if α is small). We solve this in the next strawman by considering the percentage of blocks instead of a fixed number.

Definition 9 (Percentage). *A period Pe is uncongested if x% of its blocks are not congested:* $\mathsf{UCP}_{PC}(Pe^c) = 1 \leftrightarrow \left(\sum_{b \in Pe}(1 - \mathcal{C}(b)) \geq \frac{x}{100} \cdot |Pe|\right)$.

This protocol is much more robust but has the drawback of not being monotonic. For example, if all blocks are uncongested during the first part of the period and congestion begins in the second part, then the beginning of the period is uncongested while the whole period may not be.

We now suggest the following monotonic rule:

Definition 10 (L Consecutive Blocks). *A period Pe is uncongested if there exists at least L consecutive uncongested blocks included in it:*
$\mathsf{UCP}_L(Pe^c) = 1 \leftrightarrow (\exists\ 1 \leq i \leq |Pe| - L + 1\ s.t.\ \forall\ 0 \leq j \leq L - 1 : Pe^c[i + j] = 0)$.

We show that this protocol is monotonic and inspect its efficiency in Appendix A.2. We now evaluate its robustness.

Evaluation of the robustness of the L Consecutive Blocks protocol We examine situations where the adversary attempts to manipulate the congestion signal for a given period. We separate this into two attacks: uncongestion and congestion attacks (as in Definition 4). We strive to achieve a high defense rate against both attacks, meaning finding a value L that will give a low probability for an adversary to succeed in each of the attacks separately.

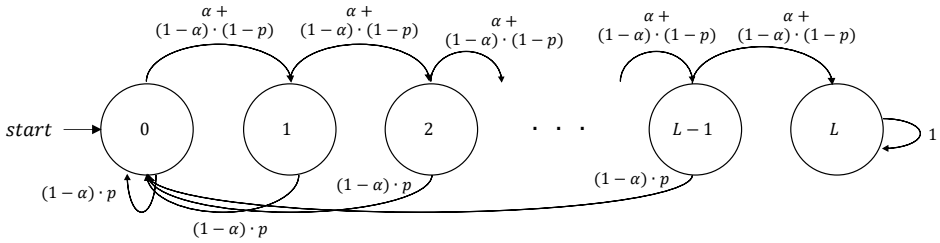
Evaluation of the uncongestion attack. In order to compute the probability of an attacker to successfully manipulate Pe into an uncongested period, we define the following matrix $T_{(L+1)\times(L+1)}$:

$$\forall\ 0 \leq i, j \leq L: \quad T_{i,j} = \begin{cases} (1-\alpha) \cdot p & \text{if } j = 0 \wedge i \neq L \\ \alpha + (1-\alpha) \cdot (1-p) & \text{if } j = i + 1 \\ 1 & \text{if } i = j = L \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

and denote by e_i the i^{th} unit vector of dimension L+1 (i.e., e_i has a 1 in the i^{th} coordinate and 0's elsewhere).

Theorem 1. *The probability of an attacker with a relative computational power α to successfully manipulate Pe into an uncongested period, in a p-congested network equals $e_1 \cdot T^n \cdot e_{L+1}^t$.*

Proof. We note that at each block, the attacker has a probability α to mine the next block, which allows them to decide its congestion level. In this context this means setting the block to be uncongested. In addition, the congestion signal of a block not mined by the attacker depends on the prevailing congestion state which is expressed by p. The probability of an honest block being congested, resp. uncongested, is hence equal to $(1-\alpha)\cdot p$, resp. $\alpha + (1-\alpha)\cdot(1-p)$. We define the following Markov chain which describes a random walk on Pe's blocks, and whose states represent the number of consecutive blocks that are uncongested at a point in time.



The initial state is 0 since it corresponds to the 0 consecutive uncongested blocks at the beginning of the walk. With each step, we move from state i to state $i+1$, for $i < L$, if the block is uncongested, and return to state 0 if it is not. If we reach state L, we stay there since it means the adversary has reached the goal of L consecutive uncongested blocks in Pe and can manipulate it to an uncongested period.

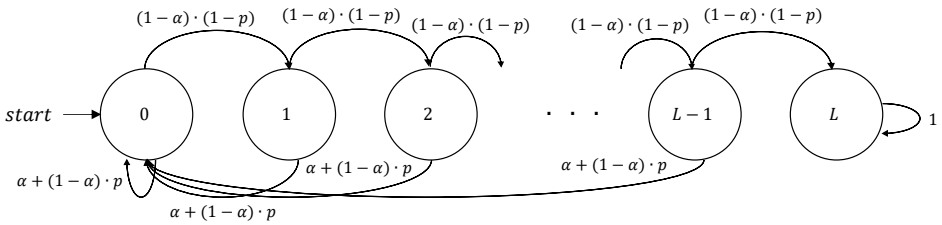
T is the corresponding transition matrix, and hence the probability of reaching state L in $|Pe| = n$ steps is expressed by $e_1 \cdot T^n \cdot e_{L+1}^t$. □

Evaluation of the congestion attack. For the attack in the opposite direction we define $\hat{T}_{(L+1)\times(L+1)}$ as follows:

$$\forall\ 0 \le i,j \le L: \quad \hat{T}_{i,j} = \begin{cases} \alpha + (1-\alpha)\cdot p & \text{if } j = 0 \wedge i \neq L \\ (1-\alpha)\cdot(1-p) & \text{if } j = i+1 \\ 1 & \text{if } i = j = L \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Theorem 2. *The probability of an attacker with a relative computational power α to successfully manipulate Pe into a congested period, in a p-congested network equals $1 - e_1 \cdot \hat{T}^n \cdot e_{L+1}^t$.*

Proof. This time, if the attacker succeeds in mining a block, they will make it congested. Therefore the probability for a block to be uncongested is $(1-\alpha)\cdot(1-p)$. As before, we define the following Markov chain whose states represent the number of consecutive blocks that are uncongested at a point in time in Pe:



$\hat{T}$ is the corresponding transition matrix. Therefore, the probability for the adversary to succeed in the congestion attack is equivalent to the probability that the rest of the miners will not reach the L state in n steps, which is expressed by: $1 - e_1 \cdot \hat{T^n} \cdot e^t_{L+1}$. □

Now that we have the attacks' success rates, we examine the robustness of the protocol against both attacks for different values of L.

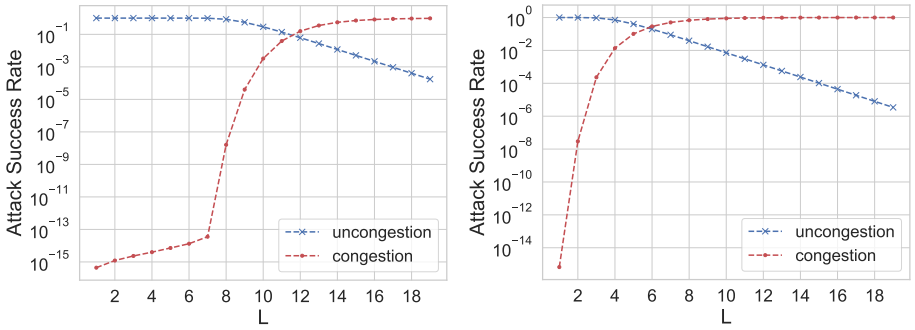
Although attacks are potentially expensive for the adversary (who needs to change the contents of its block and hence loses transaction fees), we still desire a low success probability for the attack even for strong attackers. We assume in the following evaluations that the attacker controls 33% of the computational power.

Given that congestion may cause period extension, we need a value for L that gives protection also against attacks over longer periods. We examine the behavior of the protocol for periods as long as $\hat{M}$ using different values for L.

The value p should represent realistic network conditions. For our analysis we pick $p = 0.85$ when studying the congestion attack, to simulate more congested settings or $p = 0.15$ when studying the uncongestion attack, to simulate relatively uncongested settings. Other values can be plugged in if needed for other conditions. We start by examining the robustness of the protocol for a period of 1 day.

Figures 1a-1b present the probability of success in both attacks for two different period lengths: 6450 blocks in Figure 1a and 144 blocks in Figure 1b. These periods correspond, roughly, to a single day in Ethereum (6450 blocks) and in Bitcoin (144 blocks). The red curves correspond to the congestion attack and the blue curve to the uncongestion attack. We compute these probabilities for different values of L.

The results in both figures show there is no value L that gives a probability of success less than 1% for both attacks. Formally, it shows that the L Consecutive protocol cannot be simultaneously $(0.33, 0.15, 0.01, 1\ day)$-congestion robust and $(0.33, 0.85, 0.01, 1\ day)$-uncongestion robust. Therefore, we find the L Consecutive protocol not sufficiently secure. Intuitively, this is because more robust estimates of congestion are typically obtained over longer observation windows.



(a) $|Pe| = 6450 \sim 1$ day in Ethereum (b) $|Pe| = 144 \sim 1$ day in Bitcoin

Fig. 1: Attack success rate as a function of L, for $\alpha = 0.33$

The L Consecutive protocol obtains longer observations if L is increased, but then the requirement for consecutive blocks to be uncongested is too strict and is not robust. As a result of this insight we now propose a new protocol that generalizes the L Consecutive protocol and allows for longer observation windows with a relaxed condition for uncongestion.

4.2 Sliding Window (K-out-of-N) Protocol

Definition 11 (K-out-of-N Sliding Window).
A period Pe is uncongested if there exists a period $\hat{Pe}$ of length N included in it in which at least K blocks are uncongested.

$$UCP_{SW}(Pe^c) = 1 \leftrightarrow \left(\exists\ \hat{Pe} \subseteq Pe : |\hat{Pe}| = N \wedge \left(\sum_{b \in \hat{Pe}} (1 - \mathcal{C}(b)) \geq K \right) \right)$$

We note that the L Consecutive protocol is a special case in which $L = N = K$.

Proposition 4. *The Sliding Window protocol is monotonic.*

Proof. Given an uncongested period Pe_1 according to the 'Sliding Window' protocol which is included in period Pe_2:

$$\mathsf{UCP}_{SW}(Pe_1^c) = 1 \Rightarrow \left(\exists\ \hat{Pe} \subseteq Pe_1 : |\hat{Pe}| = N \wedge \left(\sum_{b \in \hat{Pe}} \mathcal{C}(b) \geq K \right) \right)$$

$$Pe_1 \subseteq Pe_2 \Rightarrow \left(\hat{Pe} \subseteq Pe_2 \right) \wedge \left(\sum_{b \in \hat{Pe}} \mathcal{C}(b) \geq K \right)$$

$$\Rightarrow \mathsf{UCP}_{SW}(Pe_2^c) = 1$$

□

We now evaluate its efficiency.

Proof size. In order to provide evidence for the uncongestion of period Pe of size n, it is enough to point to a window in which uncongestion occurs. Formally, to present $\pi = i \in \{1, ..., n - K + 1\}$ s.t. $\sum_{l=i}^{i+N}(1 - \mathcal{C}(Pe[l])) \geq K$.

Refresh Information. Given a congested period Pe, and $\hat{Pe}$ that extends it, in order to determine the congestion level of the extended period $\mathsf{UCP}_{SW}(\hat{Pe^c})$, it is enough to check only windows that overlap blocks in $\hat{Pe} \setminus Pe$.

Evaluation of the Sliding Window protocol's robustness We consider the two attacks in Definition 4. We first note that the two attacks may differ in their consequences. While the congestion attack can cause a delay in the response deadline (i.e. a deadline will be extended even if it is not really needed), the uncongestion attack might lead participants to miss the chance to respond on time, as the deadline will not be extended even if the network is congested. The damage in each case depends on the particular use case. For example in the case of payment channels, not responding in time is more severe and may lead to financial losses. We, however, strive to achieve a high level of security against both attacks, i.e., to find values for parameters (N, K) that will yield a low probability of success in both attacks.

We begin with presenting upper bounds on the probabilities of success in each of the attacks.

Theorem 3. *The probability of an attacker with a relative computational power α to successfully manipulate Pe into an uncongested period, in a p-congested network is bounded above by $(n - N + 1) \cdot \sum_{j=K}^{N} \binom{N}{j} \cdot q^j \cdot (1 - q)^{N-j}$, for $q = \alpha + (1 - p) \cdot (1 - \alpha)$.*

Proof. The probability for a block to be uncongested during this attack is $q = \alpha + (1 - p) \cdot (1 - \alpha)$. In a period of size n, there are $n - N + 1$ different Sliding Windows. We denote by A_i the event in which there are K out of N uncongested blocks in the i^{th} sliding window. Therefore, the probability of a single sliding window being uncongested is $P(A_i) = \sum_{j=K}^{N} \binom{N}{j} \cdot q^j \cdot (1-q)^{N-j}$. To succeed in the uncongestion attack, at least one of the sliding windows has to be uncongested, which is expressed by $P(\cup_{i=1}^{n-N+1} A_i)$. We use the union bound to bound this probability and get:
$P(\cup_{i=1}^{n-N+1} A_i) \leq \sum_{i=1}^{n-N+1} P(A_i) = (n - N + 1) \cdot \sum_{j=K}^{N} \binom{N}{j} \cdot q^j \cdot (1 - q)^{N-j}$ □

Theorem 4. *The probability of an attacker with a relative computational power α to successfully manipulate Pe into a congested period, in a p-congested network is bounded above by $(\sum_{j=0}^{K-1} \binom{N}{j} \cdot q^j \cdot (1 - q)^{N-j})^{\lfloor \frac{n}{N} \rfloor}$, for $q = (1 - p) \cdot (1 - \alpha)$.*

Proof. The probability for a block to be uncongested is $q = (1-p)\cdot(1-\alpha)$. We denote by B_i the event in which there are less than K uncongested blocks in the i^{th} sliding window. The probability of a single sliding window being congested is $P(B_i) = \sum_{j=0}^{K-1}\binom{N}{j}\cdot q^j\cdot(1-q)^{N-j}$. To succeed in the congestion attack, all sliding windows in the period must be congested, which is expressed by $P(\cap_{i=1}^{n-N+1}B_i)$. We bound this probability by $P(\cap_{i=1}^{\lfloor\frac{n}{N}\rfloor}B_{N\cdot(i-1)+1})$, i.e. we consider a subset of events B_i that are independent from each other (removing overlapping windows). We compute the intersection of the pairwise independent events, and get: $P(\cap_{i=1}^{n-N+1}B_i) \leq P(\cap_{i=1}^{\lfloor\frac{n}{N}\rfloor}B_{N\cdot(i-1)+1}) = \prod_{i=1}^{\lfloor\frac{n}{N}\rfloor}P(B_{N\cdot(i-1)+1}) = (\sum_{j=0}^{K-1}\binom{N}{j}\cdot q^j\cdot(1-q)^{N-j})^{\lfloor\frac{n}{N}\rfloor}$. □

We would like to compute the robustness of the protocol for 1 day to 1 hour sliding windows. We examine the situation where a period Pe of size n is chosen at random and the blockchain is $p-$*congested* for values of $p = 0.85$ (relatively congested) and $p = 0.15$ (relatively uncongested) against an attacker with computational power $\alpha \leq 0.33$. In the evaluation we allow periods to be extended up to two weeks, a reasonable time for congestion to pass. We set $\hat{M}$-maximum extension (see definition 3) accordingly (90300 blocks in Ethereum and 2016 blocks in Bitcoin).

We first evaluate the attack over Ethereum, computing the above bounds for different sliding window sizes. We begin with a sliding window of 1 day ($N = 6450$), setting $K = \frac{N}{2} = 3225$. Figure 2 presents the two upper bounds for the different possible period lengths $N \leq n \leq \hat{M}$. For the protocol to be considered secure, we need low values in both curves for the different period lengths (since periods might be extended). As we can see the probabilities in the graph are extremely low, which shows that the protocol is very secure. We emphasize that the blue curve is not horizontal as shown in the graph, all of its values are smaller than 10^{-323}. Note that these are only upper bounds and the actual probabilities are even lower.

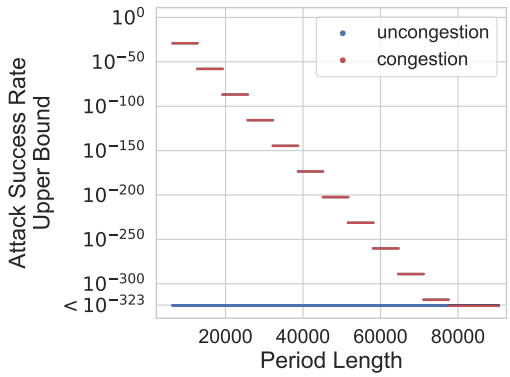


Fig. 2: Upper bounds on the attacks' success rates as a function of the period length, for $\hat{M} = 90300, N = 6450, K = 3225, \alpha = 0.33$

We evaluate the attack for smaller sliding windows. The following table summarizes our results:

N	K	Uncongestion	Congestion
6450 (1 day)	3225	$< 10^{-323}$	1.44×10^{-29}
3225 (12 hours)	1612	1.26×10^{-10}	8.06×10^{-16}
1612 (6 hours)	815	7.14×10^{-5}	1.08×10^{-7}
806 (3 hours)	421	8.87×10^{-3}	3.16×10^{-3}

The wider the sliding window is, the greater the protection. For smaller sliding windows - such as 1 hour ($N = 269$), we can achieve 99% defense rate against each attack if we lower the attackers' computation power to $\alpha \leq 0.27$ (instead of 0.33). We have provided examples of N, K values and the level of protection they provide (an upper bound), but these are configurable and subject to the user's discretion. One can choose to increase the level of protection from one attack at the expense of the other, or to set a larger initial period length ($> N$) to increase the protection.

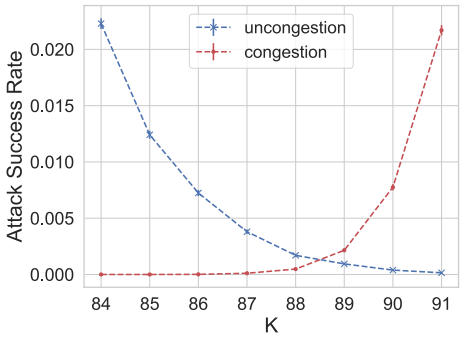
We move to examine what happens with smaller periods such as in Bitcoin which has longer block intervals. To do so, we set $\hat{M} = 2016$ and begin with a sliding window of 1 day ($N = 144$).

We use a simulation to draw 100,000 samples $Pe^c \sim B(\hat{M}, p)$ of congestion vectors and to compute the success rates of both attacks among the samples (Figures 3a,c,d). We use error plots to plot the standard error of the data however the errors are very small and therefore are almost invisible in the graphs.

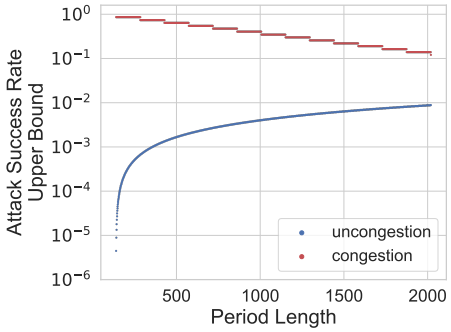
Figure 3a presents the probability of success in each of the attacks for different K values. As the graph shows, choosing $K = 89$ gives protection against both attacks. We compute the upper bounds (from Theorems 3-4) for this value of K in Figure 3b. The presented bounds as they appear in the graph are loose compared to the simulation results and give a low level of defense, especially against the congestion attack. These bounds give us useful, but non-tight, upper bounds on the results for periods that are of longer lengths, for which the probability are extremely small. To get more precise results, we use more simulations to compute the congestion attacks success rate for different period lengths and present the results in Figure 3c. We present multiple curves, each corresponding to a different value of α, the computational power of the attacker. The defense rate against congestion attacks is extremely low for short periods. For $\alpha = 0.33$, we reach a $> 99\%$ defense rate only for periods of $\sim$ 11 days or more. Lowering the computational power of the adversary naturally improves these results. For example, considering an attacker with computational power $\alpha = 0.2$ results with an above 0.9995 defense rate for period lengths starting from 2 days.

Finally, in Figure 3d we consider an attacker with a computational power $\alpha = 0.2$ and show the congestion attack success rate for different choices of N, K - which correspond to sliding windows of lengths 24/12/6/3 hours. We did not present the uncongestion attack results which had above 99% defense rate for any $N \leq n \leq \hat{M} = 2016$.

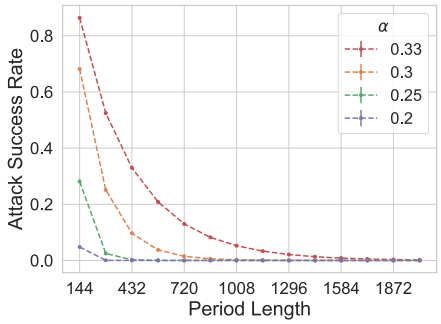
We conclude that the longer the periods are, the higher and more effective the protection against attacks is. In Ethereum we obtained very high defense rates



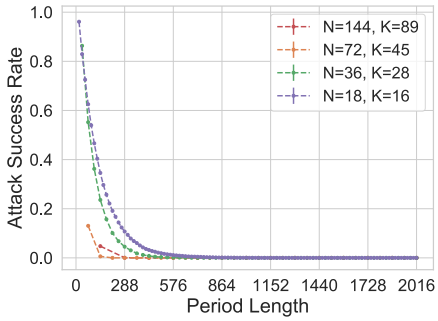
(a) Attack success rate as a function of K, for $n = 2016, N = 144, \alpha = 0.33$



(b) Upper bounds on the attacks' success rates as a function of the period length, for $N = 144, K = 89, \alpha = 0.33$



(c) Congestion attack success rate as a function of the period length, for $N = 144, K = 89$



(d) Congestion attack success rate as a function of the period length, for $\alpha = 0.2$

Fig. 3: Evaluation of the attacks' success rates for $\hat{M} = 2016$ (2 weeks in Bitcoin)

even when choosing short sliding window sizes and against strong attackers. In Bitcoin on the other hand, we need to compromise on the window size and on the attackers' power to achieve higher defense.

We defined uncongested period protocols and suggested a concrete one, the Sliding Window, which meets our requirements (as defined in Section 3.3). In the next section we will describe how to use an uncongested period protocol to adjust the challenge response protocol to deal with congested periods.

4.3 Application to Challenge-response Protocols

A challenge-response protocol consists of a challenge that takes effect at time T_c and a response deadline T_{rd} (see section 3.1). We link T_c and T_{rd} to their corresponding block height and denote by $b(T)$ the block at height T.

The parties involved in the challenge decide in advance on an uncongestion period protocol UCP to use. We recall that UCP:$\{0,1\}^* \to \{0,1\}$ accepts a

congestion vector (a binary series representing the congestion signal of blocks in a period) and returns 1 if the period is congested and 0 otherwise. To apply the uncongestion period protocol, the parties adjust T_{rd} to a short deadline which gives them a reasonable time to respond to the challenge assuming an optimal case with no congestion.

The response deadline T_{rd} is applied only in the event when the challenge window $Pe = (b(T_c), b(T_c + 1), ..., b(T_{rd}))$ is uncongested. In the case when the challenge window is congested, we repeatedly extend T_{rd}, 1 block at a time, as long as it remains congested. To avoid an edge case where the deadline is extended indefinitely, we define $\hat{T}_{rd} = T_c + \hat{M}$ an upper bound on the deadline (see definition 3). The challenge-response protocol adjustment is summarized in the algorithm below. In Appendix C, we discuss the special case of payment channels.

$T_c \leftarrow init$
$T_{rd} \leftarrow init$
$Pe = (b(T_c), b(T_c + 1), ..., b(T_{rd}))$
$Pe^c \leftarrow congestion\_vector(Pe)$
while $\mathsf{UCP}(Pe^c) = 0$ *and* $T_{rd} < \hat{T}_{rd}$ **do**
  $T_{rd} \leftarrow T_{rd} + 1$
  $Pe = (b(T_c), b(T_c + 1), ..., b(T_{rd}))$

We emphasize that the extension of the deadline is not necessarily carried out at the exact moment of the deadline (since smart contract actions need to be triggered by a transaction to the contract). Instead, a transaction that is submitted afterwards is determined to be either before or after the deadline given any possible extensions that are due. The uncongestion period protocol (UCP) is specified in advance in the smart contract, and the deadline calculation is triggered either by a late response to the challenge, or by the challenger that claims that a response did not arrive in time.

5 Implementation

We provide an implementation of the Sliding Window protocol as an Ethereum smart contract using EIP-1559 *base fee* to determine block congestion. EIP 1559 implements a *base fee* that is adjusted up and down by the protocol based on how congested the network is. The EVM supports fetching the *base fee* of the highest (current) block. We suggest extending this to fetch the *base fee* of any block, and to add an opcode that checks whether a block is congested (without such opcodes, it is not possible to fully implement the mechanisms we proposed in this paper). This opcode will receive as inputs a block and a maximum base fee (chosen by a user) and will return whether the maximum base fee exceeds the block's base fee.

In the implementation we set the sliding window size equal to the initial deadline of the examined period (before being granted any extension).

The full github[1] repository includes the smart contracts, the new opcode and tests. In addition, we include the Solidity code of the contracts in Appendix D.

6 Conclusion

In this paper we tackled a problem that arises when challenge-response protocols face congested periods. When the network experiences congestion, users will often miss the response deadline, which can lead to serious issues such as financial loss. We formalized the problem and proposed a new protocol called the Sliding Window as a solution. Our protocol defines a reliable way to detect congested periods by looking only at the data available on-chain. We then used this to extend the challenge response deadline when congestion occurs. We studied the security of the protocol for different parameters. Our results showed that it is possible to decrease the time settlement (deadline) of challenge-response protocols significantly, while expanding the security of the protocol to deal with cases of congestion.

For future work, it would be interesting to evaluate and optimize this protocol and its security analysis for more realistic congestion settings. In particular, settings in which congestion is correlated between consecutive blocks and to provide more experimental analysis of these settings. Additionally, exploring whether Ethereum's proposed base fee can be used as a sufficiently robust congestion signal is also of interest.

7 Acknowledgments

Ayelet Lotem and Aviv Zohar are partially supported by grants from the Israel Science Foundation (grants 1504/17 & 1443/21) and by a grant from the HUJI Cyber Security Research Center in conjunction with the Israel National Cyber Bureau.

References

1. Bano, S., Sonnino, A., Al-Bassam, M., Azouvi, S., McCorry, P., Meiklejohn, S., Danezis, G.: Sok: Consensus in the age of blockchains. In: Proceedings of the 1st ACM Conference on Advances in Financial Technologies. pp. 183–198 (2019)
2. Bitcoin optech: Child pays for parent (cpfp), `https://bitcoinops.org/en/topics/cpfp`
3. Mempool manipulation enabled theft of $8m in makerdao collateral on black thursday: Report, `https://www.coindesk.com/tech/2020/07/22/mempool-manipulation-enabled-theft-of-8m-in-makerdao-collateral-on-black-thursday-report/`

[1] https://github.com/stonecoldpat/slidingwindow

4. Replace by fee, https://en.bitcoin.it/wiki/Replace_by_fee
5. Buterin, V., Conner, E., Dudley, R., Slipper, M., Norden, I., Bakhta, A.: Eip-1559: Fee market change for eth 1.0 chain. URL: https://eips. ethereum. org/EIPS/eip-1559 (2019)
6. Consensys: The inside story of the cryptokitties congestion crisis (Feb 2018), https://consensys.net/blog/news/the-inside-story-of-the-cryptokitties-congestion-crisis/
7. Croman, K., Decker, C., Eyal, I., Gencer, A.E., Juels, A., Kosba, A., Miller, A., Saxena, P., Shi, E., Sirer, E.G., et al.: On scaling decentralized blockchains. In: International conference on financial cryptography and data security. pp. 106–125. Springer (2016)
8. Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., Breidenbach, L., Juels, A.: Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges. arXiv preprint arXiv:1904.05234 (2019)
9. Eyal, I., Gencer, A.E., Sirer, E.G., Van Renesse, R.: Bitcoin-ng: A scalable blockchain protocol. In: 13th {USENIX} symposium on networked systems design and implementation ({NSDI} 16). pp. 45–59 (2016)
10. Felten, E.: Fighting censorship attacks on smart contracts. https://medium.com/offchainlabs/fighting-censorship-attacks-on-smart-contracts-c026a7c0ff02 (2020)
11. Frangella, E.: Crypto black thursday: The good, the bad, and the ugly, https://medium.com/aave/crypto-black-thursday-the-good-the-bad-and-the-ugly-7f2acebf2b83, accessed: 2021-08-31
12. Gudgeon, L., Moreno-Sanchez, P., Roos, S., McCorry, P., Gervais, A.: Sok: Layer-two blockchain protocols. In: International Conference on Financial Cryptography and Data Security. pp. 201–226. Springer (2020)
13. Harris, J., Zohar, A.: Flood & loot: A systemic attack on the lightning network. In: Proceedings of the 2nd ACM Conference on Advances in Financial Technologies. pp. 202–213 (2020)
14. Kelkar, M., Zhang, F., Goldfeder, S., Juels, A.: Order-fairness for byzantine consensus. In: Annual International Cryptology Conference. pp. 451–480. Springer (2020)
15. Kursawe, K.: Wendy, the good little fairness widget: Achieving order fairness for blockchains. In: Proceedings of the 2nd ACM Conference on Advances in Financial Technologies. pp. 25–36 (2020)
16. Roughgarden, T.: Transaction fee mechanism design for the ethereum blockchain:an economic analysis of eip-1559. Tech. rep., Department of Computer Science, Columbia University (December 2020)
17. Sokolov, K.: Ransomware activity and blockchain congestion. Journal of Financial Economics (2021)
18. Sompolinsky, Y., Lewenberg, Y., Zohar, A.: Spectre: a fast and scalable cryptocurrency protocol. IACR Cryptol. ePrint Arch. **2016**(1159) (2016)
19. Sompolinsky, Y., Zohar, A.: Secure high-rate transaction processing in bitcoin. In: International Conference on Financial Cryptography and Data Security. pp. 507–527. Springer (2015)
20. Sompolinsky, Y., Zohar, A.: Phantom. IACR Cryptology ePrint Archive, Report 2018/104 (2018)
21. Tanana, D.: Avalanche blockchain protocol for distributed computing security. In: 2019 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom). pp. 1–3. IEEE (2019)

22. Wang, G., Shi, Z.J., Nixon, M., Han, S.: Sok: Sharding on blockchain. In: Proceedings of the 1st ACM Conference on Advances in Financial Technologies. pp. 41–61 (2019)

A Proofs

A.1 Proofs of Section 3.2

Proof of Proposition 1.

Proposition 1. *A miner manipulating a block* $\mathbf{B}$ *to make it* (θ_1, γ_1)*-congested when it is not will lose a potential profit of at-least* $\mathcal{B} \cdot \int_{1-(\gamma_1-\gamma_{\mathbf{B}}(\theta_1))}^{1} \theta_{\mathbf{B}}(\gamma)\, d\gamma$.

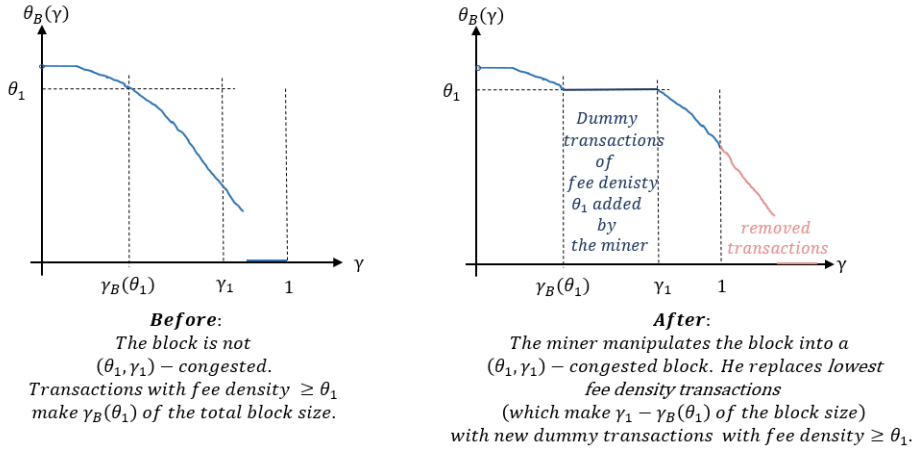


Fig. 4: Cumulative function of the θ-weight threshold of a block $\mathbf{B}$ before and after miner manipulation

Proof. Given a block $\mathbf{B}$ which is **not** (θ_1, γ_1)-congested ($\mathcal{C}_{\theta_1,\gamma_1}(\mathbf{B}) = 0$), it is possible to manipulate it into a block $\overline{\mathbf{B}}$ which is (θ_1, γ_1)-congested ($\mathcal{C}_{\theta_1,\gamma_1}(\overline{\mathbf{B}}) = 1$) by replacing some of its transactions with dummy transactions that have fee density $\geq \theta_1$. In order to maximize its revenue, the adversary will remove the transactions with the lowest fee density. The minimum portion of transactions that the adversary needs to remove is $\gamma_1 - \gamma_{\mathbf{B}}(\theta_1)$ (by the definition of $\gamma_{\mathbf{B}}$), and by doing so the miner misses the rewards associated with removing these legitimate transactions.

Using the notations from above we compute a lower bound on the miner's loss, which can be expressed by $\mathcal{U}_{\overline{\mathbf{B}}} \leq \mathcal{U}_{\mathbf{B}} - \mathcal{B} \cdot \int_{1-(\gamma_1-\gamma_{\mathbf{B}}(\theta_1))}^{1} \theta_{\mathbf{B}}(\gamma)\, d\gamma$ (see Figure 4). Note this is only a lower bound since a miner can remove transactions only in their entirety and not parts of them. □

Proof of Proposition 2.

Proposition 2. *A miner manipulating a block* $\mathbf{B}$ *to reverse its signal from* (θ_1, γ_1)*-congested to not congested will lose a potential profit of at-least* $\mathcal{B} \cdot \int_{\gamma_1}^{\gamma_{\mathbf{B}}(\theta_1)} (\theta_{\mathbf{B}}(\gamma) - \theta_1)\, d\gamma$.

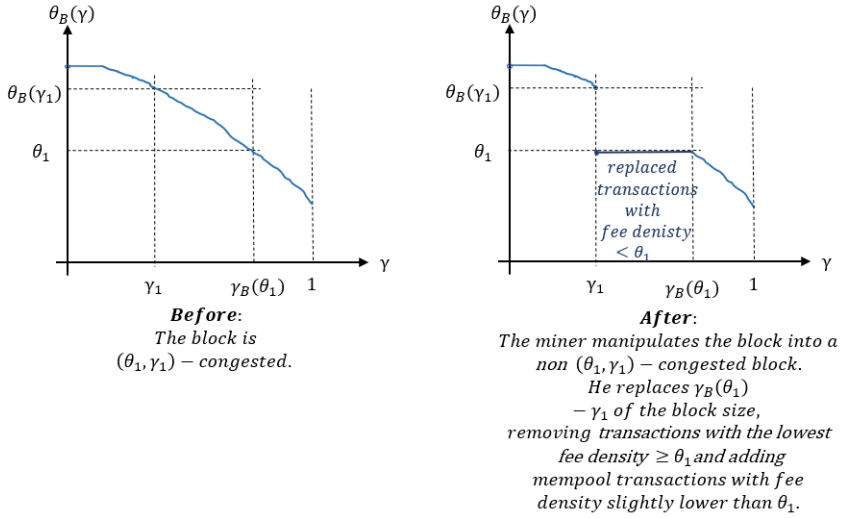


Fig. 5: Cumulative function of the θ-weight threshold of a block $\mathbf{B}$ before and after miner manipulation

Proof. The cost depends on the state of the mempool. It is likely that transactions with fee density slightly lower than θ_1 will be available in the miner's mempool allowing him to replace the original $\geq \theta_1$ fee density transactions with them and lose the fee difference. This allows us to give a bottom bound on the loss which can be expressed by $\mathcal{U}_{\overline{\mathbf{B}}} \leq \mathcal{U}_{\mathbf{B}} - \mathcal{B} \cdot \int_{\gamma_1}^{\gamma_{\mathbf{B}}(\theta_1)} (\theta_{\mathbf{B}}(\gamma) - \theta_1)\, d\gamma$ (see Figure 5). □

A.2 Proof of Section 4

Proposition 3. *The L Consecutive Blocks protocol is monotonic.*

Proof. Given a period Pe_1, uncongested according to the L Consecutive protocol, which is included in period Pe_2:

$$\mathsf{UCP}_L(Pe_1^c) = 1 \Rightarrow$$
$$\exists\ 1 \leq i_1 \leq |Pe_1| - L + 1\ s.t.\ \forall\ 0 \leq j \leq L-1 : Pe_1^c[i_1 + j] = 0$$
$$Pe_1 \subseteq Pe_2 \Rightarrow$$
$$\exists\ 1 \leq k \leq |Pe_2| - |Pe_1| + 1\ s.t.\ \forall\ 1 \leq d \leq |Pe_1| : Pe_1^c[d] = Pe_2^c[k + d - 1]$$
$$\Rightarrow for\ i_2 = k + i_1 - 1,\ \forall\ 0 \leq j \leq L-1 : Pe_2^c[i_2 + j] = 0$$
$$\Rightarrow \mathsf{UCP}_L(Pe_2^c) = 1$$

□

We now evaluate its efficiency.

Proof size. In order to provide evidence for uncongestion of period Pe of size n, it is enough to point to the location of the first block in a series of L consecutive uncongested blocks. Formally, to present $\pi = \min\{i \in \{1, ..., n - L + 1\} \mid \forall\ 0 \leq j \leq L - 1 : Pe^c[i + j] = 0\}$.

Refresh Information. Given a congested period Pe, and $\hat{Pe}$ that extends it, in order to determine the congestion level of the extended period $\mathsf{UCP}_L(\hat{Pe^c})$, it is enough to check only $\hat{Pe} \setminus (Pe[: -(L-1)])$ - the period beginning L-1 blocks before Pe ends.

B Block Congestion Definition - Examples

In definition 1 we offered to define the congestion of a block based on the fee densities. Other less effective ways in which block congestion could be defined are included below:

- *By transaction with lowest fee density:* A block is θ-congested if the lowest fee density for a transaction in it is bigger than θ: $\min_{tx \in \mathbf{B}} \phi(tx) \geq \theta$. However a miner could decide to always enter a single transaction with a very low fee density (less than θ) and easily change a block from congested to not congested, hence this scheme is easily manipulable.
- *By transaction with highest fee density:* A block is θ-congested if the highest fee density of its transactions is bigger than θ: $\max_{tx \in \mathbf{B}} \phi(tx) \geq \theta$. A miner could decide to always enter a single (dummy) transaction with a high fee density (higher than θ) and easily change a block from not congested to congested, hence this scheme is also easily manipulable.
- *By non-zero fee transaction occupancy ("block size"):* A block is γ-congested if it is full at γ-fraction of its occupancy with non-zero fee transactions: $\sum_{tx \in B:\ \phi(tx) > 0} w(tx) \geq \gamma \cdot \mathcal{B}$. A miner could artificially add transactions with positive fee density to fill the block at no cost.
- *By transaction fees (instead of fee density):* A block is (f,γ)-congested if at least a fraction γ of it is filled with transactions with fees above some value f: $\sum_{tx \in \mathbf{B}: \phi(tx) \cdot w(tx) \geq f} w(tx) \geq \gamma \cdot \mathcal{B}$. A miner could prioritize transactions by their size in order to decide the congestion signal, without lowering his profit from the fees ($\mathcal{U}_{\mathbf{B}}$).

C Payment channels

Introduction. Payment channels are a method to exchange transactions off-chain in order to reduce the on-chain load of blockchains. In this setting two parties, Alice and Bob, first lock some coins in a smart contract and can then exchange their coins without the transactions being recorded in the blockchain. They both sign and send each other transactions off-chain. Whenever one party decide to stop transacting with the other, they can close the channel by sending a specific transaction to the smart contract that will settle the final balances.

The security of payment channels relies on challenge-response protocols: if one party misbehaves by sending a wrong settlement transactions on-chain, the other party can claim the fraudulent party's fund as a compensation. For this to work, a dispute mechanism is implemented: after any party tries to settle the payment channels on-chain (challenge), the other party has a limited period of time to dispute this settlement by providing proof of wrong-doing (response). If the settlement hasn't been disputed by the response deadline, the channel is closed accordingly.

Multi-hop payments. Multi-hop payment channels are an extension to payment channels that allow users with no direct channel opened between them to transact using a route of existing channels. E.g., if both Alice and Bob have a channel opened with Charlie, they can use Charlie as a router between them to transact without opening a channel and having to lock additional funds. Conditional payments (in the form of challenge-response protocols) are often used to ensure trustless transfers. When a payment passes through a channel between Alice and Bob, they sign a conditional payment contract, which basically says that Bob can redeem the transferred amount if he presents a proof that the payment has reached its final destination (the target), before some deadline T_{rd}. This proof is released by the target of the payment once he receives it and is propagated back through the route. If the deadline T_{rd} elapses, Alice is allowed to take her money back. This mechanism ensures that Alice's coins won't be locked forever in case the proof is never presented. The deadlines T_{rd}'s of the contracts decrease along the route to avoid loss of funds. Consider for example an intermediate node, Bob, with an incoming channel contract deadline $T_{rd}(in)$ which exceeds the outgoing channel contract deadline $T_{rd}(out)$. If the response is discovered between these deadlines, Bob risks forwarding a payment that he will never receive back.

It is not enough for the deadlines to decrease, they need to be sufficiently spaced out to allow users enough time to dispute. Once Bob presents the proof to Alice, if she refuses to replace the conditional payment with an unconditional payment to him, he must reserve enough time to access the blockchain and redeem his funds. The delta $T_{rd}(in) - T_{rd}(out)$ between the deadlines is set to be greater or equal to the amount of time Bob is willing to tolerate.

The constraint of descending deadlines poses a problem with applying the Sliding Window protocol on these contracts. What if one contract along the route receives an extension while a preceding one does not? This impairs the security of the payment.

To solve this, we suggest the following adjustment. Each contract along the route will hold a challenge start time T_c, which will be set equal to the response deadline T_{rd} of the following contract in the route. That is $T_{rd}(i) = T_c(i-1)$ when i corresponds to the i_{th} channel in the route (we number the channels along the route according to the order in which the payment passes through them). In addition, each contract will hold the response deadlines of all the following contracts in the route. This allows to track extensions granted to contracts making sure that once one receives an extension, the preceding contracts will get it too. Note that once a contract is extended, for example $T_{rd}(i)$ is extended by x

blocks, the whole timeline is shifted accordingly, meaning:

$$T_{rd}(i) \leftarrow T_{rd}(i) + x \;\Rightarrow\; \forall 1 \leq j < i \;:\; \begin{cases} T_c(j) \leftarrow T_{rd}(j+1) \\ T_{rd}(j) \leftarrow T_{rd}(j) + x \end{cases} \tag{3}$$

Applying this will maintain the security of payments while allowing to shortening payments duration when the network is not congested.

D Code

```
contract BlockchainMock {

  // Simulate block.basefee(). EVM only fetches the current basefee.
  struct Block { uint baseFee; }
  Block[] public blocks;

  // Should the caller consider this block congested?
  function isCongested(uint blockNumber, uint maximumBaseFee)
  public view returns (bool) {

    if(blocks[blockNumber].baseFee > maximumBaseFee) {
      return true;
    }

    return false;
  }
}
```

```
contract SlidingWindow is BlockchainMock {

  function isPeriodCongested(uint startBlock, uint k, uint n,
    uint maximumBaseFee) public view returns (bool) {

    require(n>=k, 'N should be greater than or equal to K.');
    require(blocks.length>=n, 'Total should be greater than or equal to N.');

    uint totalCongested = 0;
    bool[] memory recordCongestion = new bool[](blocks.length - startBlock);

    for(uint i=startBlock; i < startBlock+n; i++) {
      // Keep a record of this block's congestion.
      recordCongestion[i] = isCongested(i, maximumBaseFee);

      // Sum of congestion (so far).
      if(recordCongestion[i]) {
        totalCongested = totalCongested + 1;
      }

      if(totalCongested>=k) {
        return true;
      }
    }

    // Activate the sliding window.
    for(uint i=startBlock+n; i<blocks.length; i++)

      // Remove start of the window.
      if(recordCongestion[i-n] {
        totalCongested = totalCongested - 1;
      }

      Keep a record of this block's congestion.
      recordCongestion[i] = isCongested(i, maximumBaseFee);

      // Add to the end of the window.
      if(recordCongestion[i]) {
        totalCongested = totalCongested + 1;
      }

      if(totalCongested>=k) {
        return true;
      }
  // Not congested.
  return false;
  }
}
```

```
contract Auction is SlidingWindow {

  bool start = false;
  uint startBlock;
  uint k;
  uint n;
  uint gasPriceCeiling;

  function startAuction(uint _startBlock, uint _k, uint _n,
  uint _gasPriceCeiling) public {
    startBlock = _startBlock;
    k = _k;
    n = _n;
    gasPriceCeiling = _gasPriceCeiling;
    start = true; // Kick-start the auction!
  }

  function finaliseAuction() public returns(bool) {
    if(isPeriodCongested(startBlock, k, n, gasPriceCeiling)) {
      return false;
    }

    return true;
  }
}
```