Adaptive layer-two dispute periods in blockchains

Rami Khalil
Imperial College London
rami.khalil@imperial.ac.uk

Naranker Dulay
Imperial College London
n.dulay@imperial.ac.uk

***Abstract*—Second-layer or off-chain protocols increase the throughput of permissionless blockchains by enabling parties to lock funds into smart-contracts and perform payments through peer-to-peer communication, only resorting to the smart-contracts for protection against fraud. Current protocols have fixed time periods during which participants can dispute any fraud attempts. However, current blockchains have limited transaction processing capacity, so a fixed dispute period will not always be sufficient to deter all fraudulent behaviour in an off-chain protocol. In this work, we describe how to set adaptive dispute periods that accommodate the congestion and capacity of the underlying blockchain. Adaptive dispute periods ensure that users retain the opportunity to dispute fraudulent behaviours during blockchain congestion, while increasing second-layer protocol efficiency by reducing dispute period lengths when the number of disputes is low. We describe a non-interactive argument system for setting adaptive dispute periods under the current Ethereum Virtual Machine, and discuss how to efficiently integrate built-in support for adaptive dispute periods in any blockchain. We empirically demonstrate that an adaptive-dispute second-layer protocol can handle a larger number of disputes and prevent more fraud than its non-adaptive counterparts even when users are slow to issue disputes, due to denial of service or blockchain congestion.**

I. Introduction

Second-layer cryptocurrency platforms have attracted media attention and tens of millions of US dollars worth of investments. Such platforms aim to on-board millions of users while substantially reducing the load on their underlying blockchain. This is achieved by locking user funds into a smart-contract which defines agreement terms for future withdrawal. The agreement is accompanied by a protocol that allows users to transfer ownership of portions of their locked funds to each other by only exchanging peer-to-peer messages. Users can always unlock their off-chain funds using the smart-contract, even if other users are dishonest. Such payment agreements have been realized so far in two main ways [1]: (i) two-party agreements called **payment channels**; and (ii) multi-party agreements, that we call **Plasma schemes**, where one party acts as a designated intermediary called the Plasma operator. In this paper we refer to blockchains as layer-one, and to off-chain systems as layer-two.

The main security goal in second-layer systems is to prevent fraudulent withdrawals, whereby a user attempts to unlock funds it does not own. For example, in Payment channels, a party can attempt fraud by requesting to close the channel and unlock its funds based on outdated balance information. In Plasma schemes, users can attempt to unlock their funds after having spent them [2–6], or collude with the Plasma operator to attempt to "counterfeit" their balances [7, 8] and withdraw them. To counter this, current platforms delay the completion of a withdrawal by a fixed amount of time during which users can dispute the withdrawal. After the dispute period ends, the ability to contest a withdrawal is forfeit, and the withdrawn funds are unlocked from the smart-contract into the blockchain account of the withdrawal initiator. Little attention has been given to the limitations of using fixed dispute periods.

Due to these limitations, **OmiseGo Network** platform[1] users must broadcast their transactions with very high priority to successfully protect their funds, which requires high transaction fees, while roughly no more than 300,000 **Liquidity Network** platform[2] users can be safely registered regardless of their transaction fees as of the time of writing.

As we will demonstrate, a fixed dispute period implies that only a limited number of fraud attempts can be prevented, and consequently some fraud can succeed. This limit is based on: (i) the duration of the dispute period, (ii) the transaction throughput of the blockchain, and (iii) the size of dispute transactions. Moreover, high blockchain transaction fees can hinder a user's ability to successfully publish a blockchain transaction that proves fraud before the dispute period ends. Since layer-one processing power is limited, the maximum number of disputes can be calculated using Equation 1.

$$\text{maximum disputes} = \frac{\text{block size}}{\text{dispute size}} \times \frac{\text{dispute period}}{\text{block delay}} \quad (1)$$

As a concrete example, consider the account-based Plasma scheme NOCUST [7] when deployed on the Ethereum blockchain. Once its central operator publishes a commitment of the latest state of the layer-two ledger to the smart-contract, users only have 36-hours to dispute its correctness. Given an average Ethereum block creation delay of 12 seconds between blocks, a block size of 10,000,000 gas units, a dispute cost of 360,000 gas per user, and a dispute period of 36-hours as in [7], we can calculate using Equation 1 that no more than roughly 300,000 disputes can be processed at that block size.

The rigidity of the dispute period, which has been completely unexplored prior to this paper, is problematic. First, a fixed dispute period constrains the number of users that can be safely registered in an account-based Plasma-scheme, as every user would issue a dispute in the worst case. Second, this equation generously assumes that the full transaction processing capacity of layer-one is dedicated to disputes. In

[1] https://omg.network
[2] https://liquidity.network

reality, other activity may take place and reduce the capacity to process disputes. This forces payment-channel and UTXO-based Plasma-scheme users to pay high transaction fees for high priority dispute transactions. A similar analysis can be applied to other layer-two designs.

In this paper we describe adaptive dispute cutoffs (ADC), a method for second-layer protocols to compute dynamic dispute periods that adapt in response to layer-one congestion, the number of disputes issued, and the time taken to settle the disputes.

The *requirements* for ADC are (i) a smart-contract enabled blockchain, such as Ethereum, (ii) that users possess sufficient layer-one funds for publishing disputes, (iii) that users publish disputes when necessary. The latter two requirements imply that unless a user trusts another party to launch disputes on its behalf, as in [9], a completely passive layer-two account with no blockchain balance is not granted opportunity to dispute.

Furthermore, our method does not tackle privacy, and offers no censorship resistance against powerful layer-one block proposers. The identities of parties involved in dispute transactions, and the smart-contract that implements our methods, are assumed to be public. This leaves ADC vulnerable to a powerful layer-one adversary preventing dispute transactions from being ever confirmed.

The remainder of this paper is structured as follows: Section II reviews related work. Section III overviews disputes. Section IV presents ADC. Section V introduces a backwards-compatible proving system for ADC, and Section VI discusses how to provide built-in support for ADC in layer-one blockchains. Section VII evaluates ADC, and Section VIII concludes the paper.

II. Related Work

A. Plasma

Plasma is an off-chain architecture [10] managed by a central operator. In Plasma MVP [2], the central operator periodically publishes a commitment to the state of the UTXO ledger on the parent-chain. Users must validate the full ledger with every commitment, and dispute any fake transactions or dishonestly minted funds. Plasma Cash [3] reduces the user verification overhead of Plasma MVP. Each deposit creates a new coin whose ownership can be transferred. A user withdrawing m individual coins has to initiate m disputable withdrawals using the smart-contract. Plasma Debit [4] amends Plasma Cash by adding a numeric value a to each coin to denote what portion of the corresponding deposit belongs to the owning user, while the rest is owed to the operator, bringing it closer to a payment-channel design. Plasma Prime [5] reduces the sizes of disputes. NOCUST is an account-based Plasma design [7] where an operator that withholds the data behind its commitments must be forced to reveal it within a fixed-time period. In NOCUST-ZKP [8, 11] commitments are proven in zero-knowledge to be correct, but disputes to reveal data may only be performed within a fixed amount of time. Lastly, [12] analyses the lower bound on the number of disputes that must be made in such schemes.

B. Payment channels

Channels are established between two parties and undergo three stages: (i) on-chain deployment, (ii) off-chain update and (iii) on-chain termination. Several designs secure the termination phase through the use of static-time locks [13–24], or a fixed dispute period [9, 25–37], leaving both exposed to the bandwidth limitations of layer-one. However, some designs secure their termination differently: (i) Brick [38] relies on a committee's consensus on the latest state for termination, (ii) Teechain [39] similarly uses committees called *Treasuries* that are secured by trusted hardware, (iii) and Teechan [40] uses trusted execution environments to prevent channel termination with outdated balances.

C. Transaction Metering

In the Ethereum Virtual Machine [41] (EVM), the computational, storage, and transmission efforts required to process a transaction are characterized by its *gas* consumption, where gas is a unit designed to capture the total cost of execution of a transaction. Consequently, an EVM transaction must specify both (i) a limit of how much gas the transaction may consume, and (ii) a price paid by the transaction sender per unit of gas the transaction consumes. Accordingly, the capacities of Ethereum blocks are decided by the network in terms of gas units. While some studies investigate the precision of gas as a measure of true transaction execution cost [42, 43], and others investigate how transaction gas prices can be set effectively [44], this paper, to the best of our knowledge, is the first to propose blockchain gas consumption as a measure of dispute opportunity.

III. Dispute Overview

In this section we informally define disputes, and discuss several system aspects which affect the opportunity to issue disputes in second-layer protocols.

A. Disputes

Layer-two protocols rely on a **spokesperson** publishing statements in smart-contracts about the states of layer-two data, such as accounts and payments. For example, a Plasma operator would publish a **statement** of the form: "*Commitment Y embodies the latest correct balances of all layer-two accounts*". Any errors in a statement are resolved using disputes, which are smart-contract operations that we classify as one of (i) queries, (ii) claims or (iii) proofs of fraud. A published statement that has not been disputed is only considered **final** once the statement can no longer be disputed.

Queries are requests for information about a published statement, to which a correct answer must be provided by the spokesperson in a timely manner to prevent the statement from being considered as disputed. For example, a query of the form: "*What is the layer-two balance for account X as of commitment Y?*", must be answered by returning the correct layer-two balance for the account.

Claims are incriminating allegations in the smart-contract against a statement's validity. A claim must be refuted in a

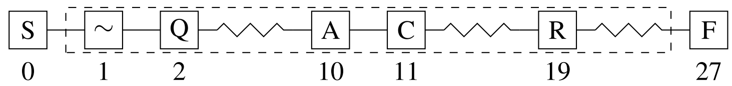

Fig. 1: Timeline of an example dispute process chronologically ordered by the block number in which each event occurs. In block 0, the spokesperson publishes a **S**tatement. No action occurs in block 1. In block 2, a user publishes a **Q**uery about the statement. In block 10, the spokesperson publishes the **A**nswer to said query. In block 11, a user publishes a **C**laim against the statement. In block 19, the spokesperson publishes a **R**efutation of said claim. In block 27, the statement is considered **F**inal. Blocks 1 to 26, surrounded by a dashed rectangle, constitute the opportunity to dispute the statement.

timely manner by proving the claim's incorrectness and the dishonesty of the claimant. For example, a claim of the form: "*The layer-two balance of account X as of commitment Y is incorrect*" can be refuted by proving that the account's layer-two balance is correctly derived. A claim should only be made when the claimant is convinced that the allegation cannot be refuted, and a refutation that shows a claim to be false should penalize the claimant.

Lastly, **proofs of fraud** are direct arguments which undeniably demonstrate that a statement is false and its spokesperson is dishonest. For example, a proof of the form: "*The spokesperson has incorrectly calculated the balance of account X as of commitment Y*" would undeniably prove that the balance is indeed incorrect, and that the operator is definitely at fault.

B. Dispute Opportunity

Following the publication of a statement, layer-two protocols are designed to grant some amount of time, or an opportunity, for users to dispute the statement. The opportunity to dispute statements of a layer-two protocol depends on two main factors, (i) the ability of users to create disputes after statement publication, and (ii) the ability to publish created disputes on the layer-one blockchain within the amount of time granted by the protocol.

1) Dispute Creation: When speaking about the ability of a user to create a dispute, we refer only to the process of a user realizing that some statement cannot (yet) safely be left to become final. This process includes creating a query about the statement, such as designating some data behind a commitment to be revealed, or constructing a claim or fraud proof against the statement. While such a process may be considered as simply a prelude to actually launching a dispute, we isolate and identify it on its own in our discussion, because it only depends on the awareness of users of published statements and answers to queries. There are two requirements to retain this ability that vary across layer-two systems:

a) Online Presence: The highest form of online presence dictates that users constantly monitor for any spokesperson activity in layer-one, as the spokesperson is free to publish statements at any time. For example, payment-channel users must always watch the blockchain for any attempts by their counter-parties to withdraw funds using outdated state information. This requires remaining constantly online, downloading every layer-one block, and checking whether there exists a transaction which attempts to close the channel. On the other hand, a lower form of online presence only requires users to come periodically online, as the spokesperson may only publish a potentially disputable statement once every pre-defined time period. For example, a NOCUST [7] operator may only publish a commitment to the state of the ledger once every pre-defined period, which means users only have to monitor the chain for a statement once per period. Consequently, the online presence requirements of a protocol constrain the opportunity to dispute statements only to users who meet a certain connectivity level to layer-one.

b) Verification Effort: Layer-two protocols require varying degrees of verification to take place by users before they can gain confidence in their disposition towards a statement. For example, when payment-channel users verify a channel closure statement, they only need to compare the balance information being used to close the channel with the latest balance information they know, often a constant-time procedure. Similarly, a NOCUST [7] user validates information proportional to how many transactions it has personally performed since the last commitment about the ledger, independent of how many transactions other users in the ledger have performed. On the other hand, Plasma MVP [2] extensively validate the entire ledger after every statement by the operator, which may require a non-trivial amount of time. Ultimately, verification requirements constrain dispute opportunity only to users who meet the necessary computational demands.

Because of a protocol's online presence and verification requirements, the opportunity available for issuing disputes can be undermined by an adversary who launches a denial of service attack that prevents users from realizing that a disputable statement is published, or by the user's own delay in learning of a statement. For example, block number 1 in Figure 1 contains no action by the user, which can be attributed to either of the aforementioned possibilities.

2) Dispute Publication: The publication of created user disputes in a layer-one blockchain largely depends on its transaction confirmation behavior. This process begins with the user broadcasting the dispute transaction to the layer-one network of block proposers, and ends when a layer-one block that contains the dispute transaction is confirmed. However, the process may end in failure if the statement being disputed had already been considered final before the dispute transaction was included in a confirmed block.

a) Gas Usage: As mentioned in Section I, only a limited number of disputes can be published per block, and consequently, dispute transactions cannot always be immediately published in a block following their creation. Notably, when publishing statements costs significantly less gas than publishing disputes, a layer-two protocol may create an asymmetry, in terms of publication power, between spokespersons and users. For example, in account-based Plasma schemes, the central operator may affect the funds of all of its users by publishing a single small commitment statement. However, to dispute this statement, all of the affected users have to issue several queries and claims, each of which is significantly more expensive than the original statement. Essentially, the efficiency of dispute publication, in terms of gas cost, in comparison to that of statement publication, determines whether an opportunity to dispute all of the effects of a statement exists.

b) Transaction Priority: The current transaction ordering mechanisms in most layer-one blockchains incentivize miners to prioritize transactions based on how much fees the transactions pay to the miners upon their execution. Such mechanisms enable wealthy transaction publishers to prioritize their transactions over others by paying higher transaction fees. Accordingly, such mechanisms enforce a minimum transaction fee for disputes, putting a price on the opportunity to dispute statements that is in line with the layer-one transaction fee market prices at the time of publication. Fundamentally, the difference in publication priority between statements and disputes affects the layer-two protocol's security determines the fairness of the dispute opportunity. For example, in a payment-channel protocol, a well-connected and wealthy spokesperson with a significant number of open channels can publish a flood of channel closure statements for all of its channels with a very high priority. Such a scenario only grants a dispute opportunity to channel users who can afford to publish disputes with higher priority than the statements of the wealthy spokesperson.

IV. Adaptive Dispute Cutoff

In this section we describe the adaptive dispute cutoff (ADC) mechanism by describing the conditions under which a statement is considered final in this model.

A. Statement Finalization

Assuming that layer-one is resistant to censorship, ADC considers gas that was unused for disputes as an indicator that potential dispute opportunities were unnecessary. For example, if after the publication of some statement in layer-one, no disputes about this statement were published in any of the blocks following the one containing the statement, then the full gas of all these blocks is considered to have added to the credibility of the statement, and is referred to as **ratification-gas**, or "***r*-gas**" for short. Consequently, the end of a statement's dispute period is determined in ADC using the amount of *r*-gas accumulated for the statement.

$$r(d) = \Delta \times \text{gps} + \alpha \times d \times c \tag{2}$$

Equation 2 defines the **required *r*-gas** for a statement to be no longer disputable as $r(d)$, a function of d, the number of disputes issued against the statement during its dispute period. The **maximum gas cost**, in gas units, for a dispute is denoted by c, while the **minimum dispute period** is denoted by Δ, a pre-defined amount of time. The **layer-one throughput**, in terms of average gas per second, is denoted by *gps*.

The **magnification factor** α adjusts the increase of the required *r*-gas remaining in response to the number of disputes issued against a statement. This allows tolerance against degradation in the online presence of users and adaptively prolongs the dispute period of a statement in response to disputes. For example, consider a layer-two protocol that requires users to be constantly online to issue disputes. If users appear online randomly during the minimum dispute period, Δ, instead of being constantly online, and a disputable statement is published, then users would not publish their disputes in perfect sequence. Instead, some amounts of *r*-gas would accumulate in between each user dispute. In turn setting the magnification factor α to a value larger than 0 would make up for the expected average gap between consecutive disputes. Furthermore, if the minimum dispute period Δ were not long enough for all possible disputes to be processed (e.g. as in NOCUST [7]), then setting the magnification factor α to a value at least 1 would allow one more user dispute to be processed for each user dispute that is processed, and allow the dispute period to be prolonged enough for all possible disputes to be issued.

In ADC, a dispute period that starts at time t_0 ends at the first point in time $t_1 > t_0 + \Delta$ if at least $r(d)$ of *r*-gas was accumulated in the layer-one blockchain between t_0 and t_1. When there are no disputes, such that $d = 0$, or the magnification factor $\alpha = 0$, the dispute period ends after the minimum dispute period Δ seconds on average. However, as disputes are issued and d increases, a delay is incurred. With some fraction, denoted by e, of the gas *not* going towards processing dispute transactions, the delay would amount to $r(d) \div ((1 - e) \times \text{gps})$. Optimizing the maximum gas cost of a dispute c can mitigate this delay.

B. Priced Statement Finalization

The risk that not all dispute transactions are published within the dispute period is a burden for a layer-two protocol's users. This risk creates an incentive to publish disputes with high fees to increase their priority, which may lead to a surge in layer-one transaction fees. The *r*-gas driven approach improves this scenario by granting users more time based on layer-one capacity. However, if layer-one fees surge, then *r*-gas would not represent an unused dispute opportunity for users with insufficient layer-one balance.

To remedy the aforementioned downside of *r*-gas, we incorporate a fixed **dispute pricing factor** p in ADC. Accordingly, the adaptive dispute period of a statement then additionally only ends when enough *r*-gas *priced less than* p, referred to as "***p*-gas**", is accumulated after the publication of the statement. Accounting for the layer-one gas pricing required for disputes allows users to plan ahead their dispute costs, and provides an equitable dispute opportunity for users who cannot afford to win a layer-one transaction fee bidding war.

Empirically, the average daily gas price in Ethereum has not risen above 1000×10^{-9} ETH as of the time of writing[3]. For Ethereum, p would determine the transaction gas price. With this, we can estimate that $p = 1000 \times 10^{-9}$ ETH would give disputes a reasonably high priority in Ethereum.

C. Constant-Price Statement Finalization

One additional concern to be addressed is the fragmentation of accumulated p-gas over several blocks. If publishing a dispute costs a significant amount of gas, then p-gas may not reflect the actual opportunity that had been available to issue such a dispute.

For example, consider a dispute which requires 10,000,000 gas to execute. If exactly 5,000,000 of gas in two consecutive blocks was accumulated as p-gas, then the total p-gas accumulated would the gas required for a single dispute (10,000,000). Relying on p-gas would lead to a false indication of a sufficient opportunity to issue the dispute, despite there being no single block containing 10,000,000 p-gas.

To remedy this, we incorporate c, the maximum gas cost of a dispute against the statement, into the ADC mechanism, such that a statement is considered final only when enough c-sized consolidated units of p-gas has been accumulated. We use the term "***c*-gas**" to refer to the accumulation of such consolidated units, which must be accumulated only in multiples of c within a single block. To elaborate, the total c-gas accumulated in the example of the two blocks in the previous paragraph, where c is equal to 10,000,000, would be zero, as no single block contained the required amount of p-gas.

D. Bounded Statement Finalization

Lastly, we set a restriction that protects ADC against potential changes in layer-one capacity. Primarily, Ethereum is known to have a dynamic gas limit, which has been increasing since its creation, and such an increase may lead to faster accumulation of c-gas that would lead to unstable minimum dispute periods. Consequently, we set an upper-bound on the amount of c-gas that maybe accumulated in each block to the pre-defined *gps* value multiplied by the expected time in between consecutive layer-one blocks. This restriction means that if the layer-one block gas limit increases, the adaptive dispute period does not end before the pre-defined minimum dispute period duration Δ.

V. Probabilistic ADC

In this section we discuss how a spokesperson can create a non-interactive computationally-sound proof [45], or non-interactive argument, to convince a smart-contract that an ADC dispute period has ended. Under this probabilistic approach, the smart-contract is first convinced, with overwhelming probability, that some p-gas was accumulated after the publication of a statement. Subsequently, the smart-contract calculates the minimum c-gas value possible for that value. If at a time $t > \Delta$ the smart-contract receives the ADC argument, and the statement had not been successfully disputed so far, the smart-contract accepts the originally published statement as final. However, the cost of verifying this argument is non-negligible, and this probabilistic approach *increases* the expected dispute period duration due to its imprecision. This approach is backwards compatible with the existing Ethereum Virtual Machine as of the time of writing.

A. Proving System

Using Algorithm 1, the spokesperson creates an argument that n random p-gas units out of the p-gas units claimed to have been accumulated are valid. The spokesperson prepares the argument as follows: **In lines 1 to 19** the spokesperson calculates the p-gas contributions of all reasonably-sized transactions following the statement. All transactions with a non-zero p-gas contribution are collected into a list sorted by order of transaction execution in the blockchain. **In line 20** the spokesperson then creates a Merkle-sum-min-max-tree commitment over this list, where the leaf weights are the p-gas contributions of each transaction, and the leaf values are (block number, transaction number) pairs, over which the minimum and maximum value annotations are calculated. Denoting the p-gas value claimed to have been accumulated by b, **in lines 21 to 35** the spokesperson generates a deterministic pseudo-random sequence of n elements from $\mathbb{Z}_b$ using any cryptographically acceptable realization of a random oracle. With b_i denoting the i^{th} value in the generated sequence, the spokesperson appends to the argument the list of opening information for the transaction which contributes the ${b_i}^{\text{th}}$ p-gas unit.

Given the argument, the smart-contract can use Algorithm 2 to derive a c-gas value from it. **In lines 1 to 18**, the smart-contract recalculates the list of b_i values using the same random oracle and commitment, and validates all the openings provided in the argument, while ensuring[4] that all opened transactions in the commitment are relatively sorted by their order of execution in the blockchain, and are published after the target statement. **In lines 19 to 20**, with some security threshold λ fixed for the smart-contract, such as $\lambda = 2^{-128}$, the smart-contract then calculates the maximum possible number of valid p-gas units that the smart-contract is confident exist in the measurement. As the argument contains no information about how the p-gas units are distributed across continuous segments, the smart-contract calculates **in lines 21 to 31** the minimum possible c-gas value that could have been accumulated in the provided blocks.

Notably, the smart-contract needs access to block and transaction information in order to verify the claimed p-gas values. In Ethereum, this can be accomplished using transaction and block inclusion proofs as done in [21], as the Ethereum Virtual Machine does not provide such built-in introspection as of the time of writing.

B. Parameterization

The two primary parameters in this scheme are (i) the number of samples in an proof, which is a dynamic parameter

[3]https://etherscan.io/chart/gasprice

[4]using the min-max tree annotations

Algorithm 1: ProvePGas

```
input :
  • maxGasPrice: p value for accumulating p-gas
  • blocks: blocks from which to accumulate p-gas
  • maxSize: maximum size of transaction to include
  • numSamples: number of samples to open
output:
  • pGas: accumulated p-gas
  • commitment: Merkle-sum commitment root
  • samples: random samples opening information
pGas ← 0;
pgValues ← [];
pgData ← [];
for block ∈ blocks do
    remGas ← block.gasLimit − block.gasUsed;
    pGas ← pGas + remGas;
    pgValues.append(remGas);
    pgData.append((block.number, -1, 0);
    for tx ∈ block.transactions do
        if |tx| > maxSize
        or tx.gasPrice > maxGasPrice then
            skip tx;
        end
        pGas ← pGas + tx.gasUsed;
        pgValues.append(tx.gasUsed);
        data ← (block.number, tx.number, tx.gasPrice);
        pgData.append(data);
    end
end
commitment, openings ←MSMMCommit(pgData,
  pgValues);
samples ← [];
for i = 0 to numSamples do
    g ← RO(commitment, pGas, i) mod pGas;
    s ← 0;
    t ← Φ;
    for j = 0 to |pgValues| do
        if s + pgValues[j] ≥ g then
            t ← (pgData[j], openings[j]);
            break;
        else
            s ← s + pgValues[j];
        end
    end
    samples.append(t);
end
return pGas, commitment, samples
```

Algorithm 2: VerifyCGas

```
input :
  • maxGasPrice: p value for accumulating p-gas
  • segmentSize: c value for accumulating c-gas
  • blocks: blocks from which to accumulate p-gas
  • pGas: accumulated p-gas
  • commitment: Merkle-sum commitment root
  • samples: random samples opening information
  • λ: probability of error
  • α: p-gas fraction claimed
output:
  • cGas: proven c-gas
n ← |samples|;
for i = 0 to n do
    g ← RO(commitment, pGas, i) mod pGas;
    weightPrefix, weightLeaf, data ←
      Open(commitment, samples[i]);
    blockNum, txNum, gasPrice ← data;
    block ← getBlock(blockNum);
    remGas ← block.gasLimit − block.gasUsed;
    tx ← getTransaction(blockNum, txNum);
    if g < weightPrefix
    or weightPrefix + weightLeaf < g
    or gasPrice > maxGasPrice
    or (txNum = −1 and remGas ≠ weightLeaf)
    or tx.gasPrice ≠ gasPrice
    or tx.gasUsed ≠ weightLeaf
     then
        abort
    end
end
if α^n > λ then
    abort
end
pgProven ← α × pGas;
incompleteSegments ← ⌊pgProven / (segmentSize−1)⌋;
remainder ← pgProven mod (segmentSize − 1);
m_1 = |blocks| × (⌊incompleteSegments / |blocks|⌋ − 1);
m_2 = incompleteSegments mod |blocks|;
if m_1 ≤ 0 then
    return 0
else if remainder = 0 then
    return m_1 + m_2
else
    return m_1 + m_2 + 1
end
```

that the spokesperson can decide per argument, and (ii) the security level λ, which dictates the level of confidence the smart-contract must have in the argument. In this section we describe the relationship between these two parameters and how they affect the percentage of *p*-gas the smart-contract can be confident to have been accumulated.

Interestingly, by setting ”maxSize” as input to Algorithm 1, the spokesperson can control the maximum cost of verification for each sample, including transaction membership proofs, and can consequently derive the maximum cost of verification of the argument in the smart-contract using the information in

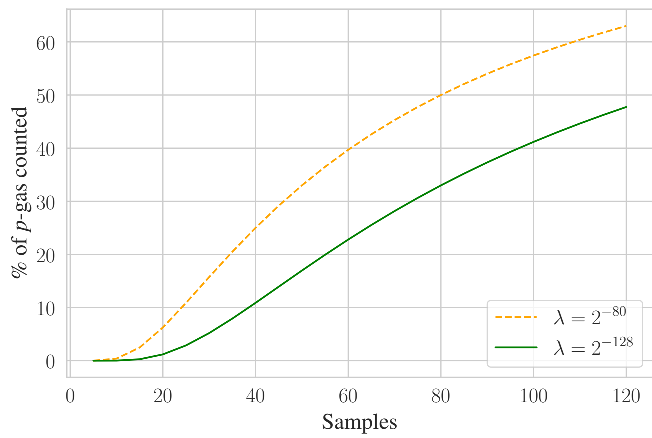

Fig. 2: Number of correct samples provided in a proof versus the percentage of p-gas that may be assumed to exist with probability $1 - \lambda$.

Figure 2. This allows the spokesperson to perform a trade-off between how much it is willing to wait for more p-gas to accumulate, and how much gas it is willing to spend to prove to the smart-contract that a certain portion of the p-gas that has been accumulated exists.

For example, by limiting the maximum cost of verification of a single sample to 100,000 gas, the spokesperson can include any transaction of size 1 kilobyte or less in its argument, and by including 40 samples, it can control the argument verification cost to be on the order of 4,000,000 gas. Smart-contracts with $\lambda = 2^{-80}$ would accept roughly 25% of the claimed p-gas value, while those with $\lambda = 2^{-128}$ would only accept roughly 10%.

VI. Native ADC

In this Section, we discuss how ADC can be efficiently enabled in layer-one with minimal overhead using a Binary Indexed Tree [46] (BIT), which is an efficient data-structure for updating and querying a subset of a d-dimensional matrix of n^d elements in $O(\log^d n)$. For the remainder of this section, we define $\texttt{upper}(\mathbb{X}, x)$, where $\mathbb{X} \subseteq \mathbb{N}$ and $x \in \mathbb{N}$, as the largest value in $\mathbb{X}$ that is less than or equal x. Similarly, $\texttt{lower}(\mathbb{X}, x)$ is the smallest value in $\mathbb{X}$ greater than or equal x.

To keep track of p-gas, a BIT can be used as follows: Let $\mathbb{P}$ be the set of indexed prices, and n the number of blocks indexed Let $\mathcal{T}_{\mathbb{P}}$ denote the two-dimensional BIT constructed over the sparse matrix of $n \times |\mathbb{P}|$ elements.

- To record that a transaction has used b units of gas at price p in block i, let $\rho = \texttt{lower}(\mathbb{P}, p)$, and increment $\mathcal{T}_{\mathbb{P}}(i, \rho)$ by b.
- To record that block i contains b unused gas units, increment $\mathcal{T}_{\mathbb{P}}(i, 0)$ by b.
- To query how much p-gas was accumulated between blocks i and j, let $\rho = \texttt{upper}(\mathbb{P}, p)$, and query $\mathcal{T}_{\mathbb{P}}$ for the subset sum between indices (i, ρ) and (j, ρ).

All operations are in $O(\log n \times \log |\mathbb{P}|)$.

To keep track of c-gas, the p-gas BIT $\mathcal{T}_{\mathbb{P}}$ is first computed, then a separate set of BITs is populated as follows: Let $\mathbb{C}$ be the set of indexed segment sizes. Let $\mathcal{T}_{\mathbb{P}}^{\mathbb{C}}$ denote the set of $|\mathbb{C}|$ two-dimensional BITs constructed, for each $c \in \mathbb{C}$, over the sparse matrices of $n \times |\mathbb{P}|$ elements each. We denote by $\mathcal{T}_{\mathbb{P}}^{c}$ the BIT for counting segments of size c.

- To record how much c-gas was accumulated in a block, after updating $\mathcal{T}_{\mathbb{P}}$ using the transactions from block i, each $\mathcal{T}_{\mathbb{P}}^{c}$ is updated as follows: Set $\texttt{temp} \leftarrow 0$. For each $p \in \mathbb{P}$ in non-decreasing order, set $\mathcal{T}_{\mathbb{P}}^{c}(i, p)$ to $c \times \lfloor \frac{\mathcal{T}_{\mathbb{P}}(i,p)}{c} \rfloor - \texttt{temp}$ and $\texttt{temp}$ to $\mathcal{T}_{\mathbb{P}}^{c}(i, p)$. This update process is in $O(|\mathbb{C}| \times |\mathbb{P}| \times \log n \times \log |\mathbb{P}|)$.
- To query how much c-gas was accumulated between blocks i and j, let $\rho = \texttt{upper}(\mathbb{P}, p)$, and query $\mathcal{T}_{\mathbb{P}}^{c}$ for the subset sum between indices (i, ρ) and (j, ρ) in $O(\log n \times \log |\mathbb{P}|)$.

The efficiency and usability of this layer-one indexing technique largely depends on the sets $\mathbb{P}$ and $\mathbb{C}$. Using exponentially increasing values, or some other small set of values, would result in efficient updates and queries. However, such coarse-grained indexing may not be perfectly suited to every application.

VII. Evaluation

In this section we evaluate our prototype ADC implementation, focusing on the cost and effectiveness of ADC in protecting layer-two protocol users. The prototype implementation is in Solidity 0.6.6 and JavaScript, and is open-source[5]. Measurements are sampled on a locally deployed test network using Ganache[6].

a) Cost: Figure 3 shows the smart-contract gas costs for ADC proof verification versus number of samples provided in a proof. The samples used consist of basic Ethereum transfers, along with their inclusion proofs. It costs roughly 500,000 gas per such additional sample. Furthermore, it costs roughly 490,005 per 256 blocks that pass in a dispute period to commit their hashes. These high costs are mostly due to the lack of transaction inspection support, as the remainder of the verification steps are inexpensive.

b) Effectiveness: In Figure 4 we plot the percentage of successful fraud attempts under ADC, compared to the layer-two systems, NOCUST and OmiseGO, with static-dispute periods. We simulate $n =$ 1,000,000 registered users with a single layer-two operator, and an Ethereum layer-one blockchain where blocks have a limit of 10,000,000 gas units, and are produced once every 20 seconds. The variable $\mathcal{R}_S$ represents the average rate per block at which the spokesperson publishes statements in the smart-contract, and $\mathcal{R}_D$ represents the average rate per block at which these statements are disputed. We run the simulation with a fixed dispute period of $\Delta =$ 24 hours for NOCUST and OmiseGO, and use the same Δ as the minimum dispute period for our ADC ledger.

[5]https://github.com/rami-github/adaptive-dispute-cutoffs
[6]https://github.com/trufflesuite/ganache

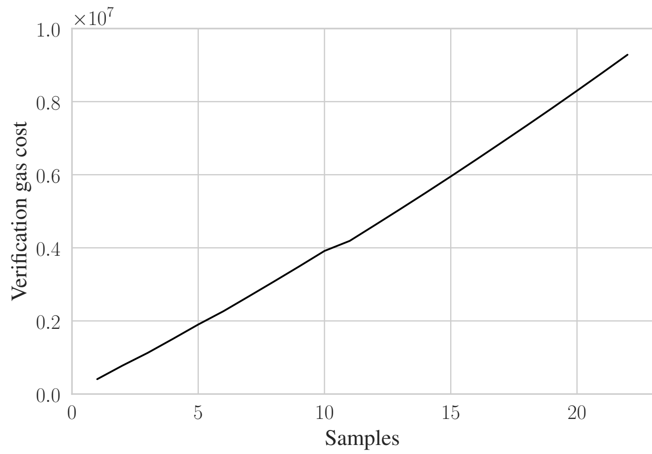

Fig. 3: Number of correct samples provided in a proof versus the gas cost of executing our verifier contract.

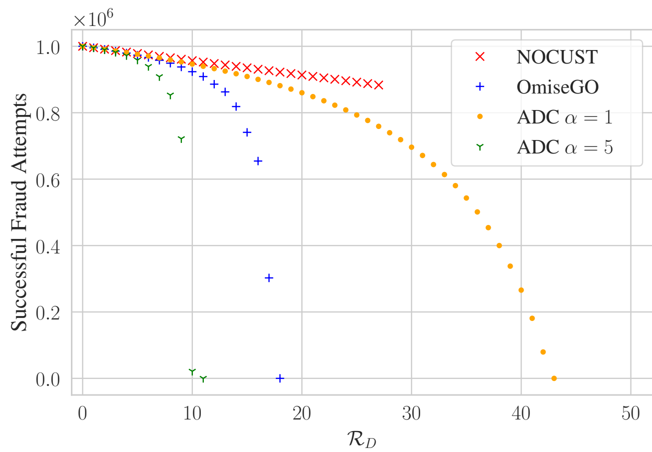

Fig. 4: Plot of estimated successful fraud attempts for 1,000,000 attempts under different $\mathcal{R}_D$ values in our ADC enabled ledger, NOCUST and OmiseGO. NOCUST permits the operator to perform a high amount of fraud due to its overeliance on static-time queries, while OmiseGO is inflexible with how slowly withdrawals can be disputed.

For our ADC enabled ledger, and NOCUST, $\mathcal{R}_S = n$ in the first block, and $\mathcal{R}_S = 0$ thereon after, as the spokesperson can publish a single commitment statement, which simultaneously affects all registered user accounts at once. For NOCUST, we consider a maximum $\mathcal{R}_D = 27$, which is the maximum rate whereby all blocks are filled with queries and their responses. For our ADC enabled ledger, we consider a maximum $\mathcal{R}_D = 52$, as at most 52 claims and their refutations can fit inside a block, while no queries are required.

In payment-channels and UTXO-based Plasma-schemes, fraudulent channel closures and fraudulent withdrawals can only be attempted per channel or transaction output respectively. Therefore, we examine the resulting successful fraud attempts while considering a maximum $\mathcal{R}_S = 29$ based the cost for a standard withdrawal initialization being on the order of an average 336,000 gas in the UTXO Plasma-based OmiseGO Network as of the time of writing. We also use a maximum value of $\mathcal{R}_D = 49$ for the OmiseGo platform, since its challenge cost is on the order of an average 200,000 gas at the time of writing. As both ongoing withdrawal initialization and their challenges share the layer-one bandwidth, we vary the value of $\mathcal{R}_D$ and derive the value of $\mathcal{R}_S$ based on the remaining bandwidth.

The main observation about our ADC ledger in Figure 4 is the impact of the two different values of α on the estimated successful fraud attempts. We then see that $\alpha = 5$ requires $\mathcal{R}_D = 10$ for zero fraud, while $\alpha = 1$ requires a higher user dispute rate of $\mathcal{R}_D = 42$.

VIII. Conclusion

We have proposed ADC, a technique to increase the robustness of second-layer protocols and explained how second-layer protocols can make use of ADC to secure disputes. We evaluated the efficacy of ADC in allowing a Plasma scheme to secure 1,000,000 accounts while providing an equitable dispute opportunity under layer-one congestion.

A. Future Work

a) Privacy and Censorship: While we do not address privacy or layer-one censorship resilience, it would be a valuable contribution to design a system where disputes cannot be easily identified or targeted for censorship.

b) Verification Costs: Using succinct zero-knowledge proving systems, such as zkSNARKS, the maximum percentage of *p*-gas that can be proven to exist could be increased through reducing the costs associated with running the verification procedure in a smart-contract. It would be a promising avenue of work to explore such designs and demonstrate their tradeoffs.

Acknowledgment

This work is supported by the Imperial College London President's PhD Scholarship.

References

[1] L. Gudgeon, P. McCorry, P. Moreno-Sanchez, A. Gervais, and S. Roos, "Sok: Off the chain transactions," *IACR Cryptology ePrint Archive*, 2019.

[2] V. Buterin, "Minimal viable plasma," https://ethresear.ch/t/minimal-viable-plasma/426, 2018, online; Accessed 03-June-2020.

[3] ——, "Plasma cash: Plasma with much less per-user data checking," https://ethresear.ch/t/plasma-cash-plasma-with-much-less-per-user-data-checking/1298, 2018, online; Accessed 03-June-2020.

[4] D. Robinson, "Plasma debit: Arbitrary-denomination payments in plasma cash," https://ethresear.ch/t/plasma-debit-arbitrary-denomination-payments-in-plasma-cash/2198, 2018, online; Accessed 03-June-2020.

[5] I. Gulamov, "Plasma prime design proposal," https://ethresear.ch/t/plasma-prime-design-proposal/4222, 2018, online; Accessed 03-June-2020.

[6] "Learn plasma," https://www.learnplasma.org, online; Accessed 03-June-2020.

[7] R. Khalil, "NOCUST-A non-custodial 2nd-layer blockchain payment hub," Master's thesis, Swiss Federal Institute of

Technology, Zurich, 2018, https://pub.tik.ee.ethz.ch/students/2018-FS/MA-2018-24.pdf.

[8] R. Khalil, P. Moreno-Sanchez, A. Zamyatin, A. Gervais, and G. Felley, "Commit-chains: Secure, scalable off-chain payments," Cryptology ePrint Archive, Report 2018/642, Tech. Rep., 2019.

[9] P. McCorry, S. Bakshi, I. Bentov, S. Meiklejohn, and A. Miller, "Pisa: Arbitration outsourcing for state channels," in *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, 2019, pp. 16–30.

[10] J. Poon and V. Buterin, "Plasma: Scalable autonomous smart contracts," *White paper*, 2017.

[11] R. Khalil, A. Gervais, and G. Felley, "NOCUST-A securely scalable commit-chain," Cryptology ePrint Archive, Report 2018/642, Tech. Rep., 2018.

[12] S. Dziembowski, G. Fabianski, S. Faust, and S. Riahi, "Lower bounds for off-chain protocols: Exploring the limits of plasma," Cryptology ePrint Archive, Report 2020/175., Tech. Rep., 2020.

[13] C. Decker and R. Wattenhofer, "A fast and scalable payment network with bitcoin duplex micropayment channels," in *Symposium on Self-Stabilizing Systems*. Springer, 2015, pp. 3–18.

[14] S. Werman and A. Zohar, "Avoiding deadlocks in payment channel networks," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer, 2018, pp. 175–187.

[15] E. Heilman, S. Lipmann, and S. Goldberg, "The arwen trading protocols," 2019.

[16] G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, and M. Maffei, "Anonymous multi-hop locks for blockchain scalability and interoperability." in *NDSS*, 2019.

[17] E. Tairi, P. Moreno-Sanchez, and M. Maffei, "A2l: Anonymous atomic locks for scalability and interoperability in payment channel hubs," Cryptology ePrint Archive, Report 2019/589, Tech. Rep., 2019.

[18] D. Hosp, T. Hoenisch, P. Kittiwongsunthorn *et al.*, "Comit-cryptographically-secure off-chain multi-asset instant transaction network," *arXiv preprint arXiv:1810.02174*, 2018.

[19] D. Piatkivskyi and M. Nowostawski, "Split payments in payment networks," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer, 2018, pp. 67–75.

[20] Y. Zhang, Y. Long, Z. Liu, Z. Liu, and D. Gu, "Z-channel: Scalable and efficient scheme in zerocash," *Computers & Security*, pp. 112–131, 2019.

[21] A. Zamyatin, D. Harz, J. Lind, P. Panayiotou, A. Gervais, and W. Knottenbelt, "Xclaim: Trustless, interoperable, cryptocurrency-backed assets," *IEEE Security and Privacy. IEEE*, 2019.

[22] G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, and S. Ravi, "Concurrency and privacy with payment-channel networks," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 455–471.

[23] E. Heilman, L. Alshenibr, F. Baldimtsi, A. Scafuro, and S. Goldberg, "Tumblebit: An untrusted bitcoin-compatible anonymous payment hub," in *Network and Distributed System Security Symposium*, 2017.

[24] C. Decker, R. Russell, and O. Osuntokun, "eltoo: A simple layer2 protocol for bitcoin," *White paper: https://blockstream.com/eltoo. pdf*, 2018.

[25] R. Khalil and A. Gervais, "Revive: Rebalancing off-blockchain payment networks," *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017.

[26] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," https://lightning.network/lightning-network-paper.pdf, 2016.

[27] M. M. Chakravarty, S. Coretti, M. Fitzi, P. Gazi, P. Kant, A. Kiayias, and A. Russell, "Hydra: Fast isomorphic state channels," Cryptology ePrint Archive, Report 2020/299, Tech. Rep., 2020.

[28] C. Buckland and P. McCorry, "Two-party state channels with assertions," in *International Conference on Financial Cryptography and Data Security*. Springer, 2019, pp. 3–11.

[29] E. Wagner, A. Völker, F. Fuhrmann, R. Matzutt, and K. Wehrle, "Dispute resolution for smart contract-based two-party protocols," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2019, pp. 422–430.

[30] A. R. Pedrosa, M. Potop-Butucaru, and S. Tucci-Piergiovanni, "Scalable lightning factories for bitcoin," in *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, 2019, pp. 302–309.

[31] P. McCorry, C. Buckland, S. Bakshi, K. Wüst, and A. Miller, "You sank my battleship! a case study to evaluate state channels as a scaling solution for cryptocurrencies," in *International Conference on Financial Cryptography and Data Security*. Springer, 2019, pp. 35–49.

[32] M. Dong, Q. Liang, X. Li, and J. Liu, "Celer network: Bring internet scale to every blockchain," *arXiv preprint arXiv:1810.00037*, 2018.

[33] S. Dziembowski, L. Eckey, S. Faust, J. Hesse, and K. Hostáková, "Multi-party virtual state channels," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2019, pp. 625–656.

[34] S. Dziembowski, S. Faust, and K. Hostáková, "General state channel networks," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 949–966.

[35] S. Dziembowski, L. Eckey, S. Faust, and D. Malinowski, "Perun: Virtual payment hubs over cryptocurrencies," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 106–123.

[36] H. Kalodner, S. Goldfeder, X. Chen, S. M. Weinberg, and E. W. Felten, "Arbitrum: Scalable, private smart contracts," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 1353–1370.

[37] A. Miller, I. Bentov, S. Bakshi, R. Kumaresan, and P. McCorry, "Sprites and state channels: Payment networks that go faster than lightning," in *International Conference on Financial Cryptography and Data Security*. Springer, 2019, pp. 508–526.

[38] G. Avarikioti, E. K. Kogias, and R. Wattenhofer, "Brick: Asynchronous state channels," *arXiv preprint arXiv:1905.11360*, 2019.

[39] J. Lind, O. Naor, I. Eyal, F. Kelbert, P. R. Pietzuch, and E. G. Sirer, "Teechain: Reducing storage costs on the blockchain with offline payment channels," in *Proceedings of the 11th ACM International Systems and Storage Conference, SYSTOR 2018, HAIFA, Israel, June 04-07, 2018*, 2018, p. 125.

[40] J. Lind, I. Eyal, P. Pietzuch, and E. G. Sirer, "Teechan: Payment channels using trusted execution environments," *arXiv preprint arXiv:1612.07766*, 2016.

[41] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, 2014.

[42] T. Chen, X. Li, Y. Wang, J. Chen, Z. Li, X. Luo, M. H. Au, and X. Zhang, "An adaptive gas cost mechanism for ethereum to defend against under-priced dos attacks," in *International Conference on Information Security Practice and Experience*. Springer, 2017, pp. 3–24.

[43] D. Perez and B. Livshits, "Broken metre: Attacking resource metering in evm," *arXiv preprint arXiv:1909.07220*, 2019.

[44] S. M. Werner, P. J. Pritz, and D. Perez, "Step on the gas? a better approach for recommending the ethereum gas price," *arXiv preprint arXiv:2003.03479*, 2020.

[45] S. Micali, "Cs proofs," in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, 1994, pp. 436–453.

[46] P. M. Fenwick, "A new data structure for cumulative frequency tables," *Software: Practice and experience*, vol. 24, no. 3, pp. 327–336, 1994.