[This is cross-posted from Solutions to Delay Attacks on Rollups | by Offchain Labs | Dec, 2022 | Medium to enable discussion here.]

One of the subtle issues for designers of rollup protocols is how to deal with delay attacks. In this post I'll talk about what they are and how Arbitrum protects against them, leading up to some exciting new developments.

Delay attacks

are malicious actions that try to prevent a rollup protocol from making progress. They don't attack the safety of the protocol, that is, they don't try to force confirmation of an incorrect result. Instead, they attack the liveness

of the protocol by trying to prevent or delay confirmation of any results at all.

These issues can be subtle, and to be honest, protocol designers often don't like to talk about them. But every Layer 2 system, whether optimistic, ZK, or other, needs to cope with questions about delay and protocol progress.

This article digs into the delay attack problem, and discusses how it was handled in various versions of the Arbitrum rollup protocol.

What are delay attacks?

In a delay attack, a malicious party (or group of parties) act within the rollup protocol, following a strategy that is designed to prevent or delay the confirmation of results back to the L1 chain.

This differs from a denial of service attack, in which the attacker tries to prevent any action from being taken in the protocol. By contrast, in a delay attack actions continue to occur, but the attacker acts in ways that prevent or delay the confirmation of results (that is, delay withdrawals of assets to L1) and force honest validators to burn gas.

Any plausible rollup protocol will require actors to stake, so that a delay attacker will necessarily lose one or more stakes. We will assume here that the attacker is willing to sacrifice stakes, within some limits, in pursuit of their attack.

We will also assume, conservatively, that the attacker has an advantage in getting their transactions on-chain, so that whenever the attacker is racing against honest parties to get a transaction on-chain first, the attacker will always win.

Finally, we will assume that an attacker can censor access to the underlying L1 chain to exclude rollup transactions, but only for a limited period, which we call the "challenge period." In particular, the attacker can enable and disable a notional "censorship mode." When censorship mode is enabled, the attacker has complete control over which transactions can reach the L1. However, the attacker can only have censorship mode enabled for a total of one challenge period. (We assume that any set of censorship-mode periods adding up to more than one challenge period will trigger a social response from the L1 community to thwart the censorship attempt.)

Evaluating protocols against delay attack

In evaluating protocols, we can ask six questions:

1. Does the protocol have a fraud proof mechanism at all? (If not, delay attacks are moot because participants can't delay confirmation of any results — not even fraudulent ones.)
2. Is there a central operator or prover who can prevent progress by simply stopping, or by withholding data? If so, that party can cause an infinite delay.
3. Does the protocol provide a trustless guarantee of eventual progress? In other words, is it the case that a single honest participant can force eventual progress, regardless of what attackers do?
4. If the protocol guarantees trustless progress, what is the upper bound on delay that an attacker can cause?
5. How does the attacker's cost scale with the length of delay caused?
6. How does the total cost of response by honest parties scale?

With those criteria established, let's evaluate two historical versions of the Arbitrum rollup protocol.

Protocol 1: Academic paper protocol

The 2018 academic paper about Arbitrum used roughly the following protocol (ignoring a unanimous mode that isn't relevant here). Any staker could assert a proposed result, which we'll call an assertion. During a time window, any subset of the other stakers could challenge the assertion, and the asserter would have to defend their assertion against each challenger, one

challenger at a time. At the end of each challenge, the losing party would forfeit their stake.

(Note that it is necessary to allow multiple stakers to oppose the assertion, and to give every challenger its own opportunity to defeat the assertion. This is necessary because a malicious party can deliberately lose a challenge that they “should” have won. Giving every challenger a separate challenge ensures that one honest challenger can defeat an incorrect assertion, no matter how many malicious parties deliberately lose challenges.)

If there was no challenge, or if the asserter won every challenge, then the assertion would be confirmed and the protocol would move forward. But if the asserter lost any of the challenges, its assertion would be rejected and the protocol state would roll back to the state before the assertion was made.

Evaluation

This protocol has working fraud proofs. This protocol does not guarantee progress, because a malicious participant can endlessly make incorrect assertions, sacrificing a stake each time but leading to an endless cycle of the same assertion being made and rejected, leading to continuous rollbacks and lack of progress.

Protocol 2: fork-and-prune

The current Arbitrum protocol — which has been deployed on every version of Arbitrum since 2020— improves the previous protocol by introducing branching. The idea is to allow multiple stakers to make competing assertions, and to treat competing assertions as forking the chain. A series of challenges then pits the branches against each other and eventually prunes off all but one branch, allowing the one remaining branch to be confirmed.

Timekeeping works as follows. Each rollup block in the chain tracks the timestamp when its first child (i.e., successor) was created by a staker. Other stakers can create additional children. Each child assertion implicitly claims that all of its older siblings are incorrect.

The staker who creates an assertion is required to stake on it, and other stakers have the option of also staking on it.

If two stakers are staked on assertions that are siblings, and neither of the two stakers is already in a challenge, then a challenge can be initiated between the two stakers, where the staker on the older of the two siblings is defending the correctness of that older sibling, while the other staker is challenging that correctness. The staker who loses the challenge forfeits their stake and is removed from the set of stakers.

This protocol includes deadlines for action. First, the deadline for creating child assertions of a parent is one challenge period after the first child has been created. Second, the deadline for staking on the an assertion is one challenge period after that assertion is created.

If the deadline for staking on an assertion has passed, and there are no stakers staked on that assertion, the assertion is pruned off. (One or more stakers must have been staked there in the past, but all of them must have lost challenges.). Any children, grandchildren, or other descendants of the pruned assertion are also pruned away at the same time.

If an assertion is older than one challenge period and has no unpruned siblings, that assertion can be confirmed, representing progress in the protocol.

Evaluation

This protocol has working fraud proofs. There is no central operator or prover who can prevent progress, so any participant can force progress. To force progress, an honest staker can post a correct child, if one does not already exist. After that, a limited time will pass before a deadline ensures that no more siblings can be created and no more stakers can stake on the siblings that exist. From that point on, the honest staker will engage in a series of challenges, defeating and removing parties staked incorrectly, one party at a time. (If multiple honest parties are staked, they can defeat adversaries in parallel.) Once all such parties are removed, the correct child assertion can be confirmed.

The most effective delay attack against this protocol has one malicious staker stake on an incorrect sibling, and N-1 malicious stakers stake on the correct sibling. No matter how many honest stakers are staked on the correct sibling, the attacker will be able to arrange that the incorrectly staked staker engages in challenges against its confederates before engaging in challenges against honest parties. (This assumes, pessimistically, that the attacker can always get in transactions before the honest parties can.) The confederates will also deliberately lose their challenges, taking as long as possible to do so. (Due to the delay rules, this could take as long as one or two challenge periods per confederate.). Only after all N-1 confederates had sacrificed themselves would the incorrectly staked staker be required to fight a challenge against an honest party, which the honest party would win, finally eliminating the incorrectly staked staker.

This last attack achieves a delay of roughly N challenge periods, at a cost to the attacker of N stakes.

The cost to honest parties, against this attack, is linear in the number of honest parties, because every honest party would

need to stake before the staking deadline.

Summary:

- guarantees progress
- cost of attack is linear in delay caused
- cost to honest parties is linear in number of honest parties

Permissioned validation

The delay attack characteristics of Protocol 2, which is currently deployed on Arbitrum, are the reason we have chosen to temporarily limit the validator role to a permissioned set of parties rather than making validation fully permissionless. Taking the last step to permissionless validation requires a protocol version that is maximally resistant to delay attacks. And you've probably guessed already that such a protocol version exists and we're currently working on implementing it.

Coming soon: Protocol 3

This post is already too long, so I'm not going to give the full download on the new protocol — that will be a future post.

But here's the gist. The next version of the Arbitrum protocol makes some small but (I think) elegant changes to how assertions and challenges work, so that the worst-case delay attack can cause only one challenge period of delay, no matter how many stakes the attackers are willing to lose.

This is based on a technical breakthrough from the Arbitrum research team that makes all-against-all challenges feasible and efficient. This allows a single honest staker to efficiently defeat an army of attackers who have posted a forest of malicious branching assertions.

The new protocol is fully specified and is being implemented now. Of course we won't roll it out on mainnet until it's thoroughly tested and the code is audited.

In a future post I'll deep-dive into the new protocol.