L2 Sequencer Uptime Feeds

Optimistic rollup protocols move all execution off the layer 1 (L1) Ethereum chain, complete execution on a layer 2 (L2) chain, and return the results of the L2 execution back to the L1. These protocols have asequencer that executes and rolls up the L2 transactions by batching multiple transactions into a single transaction.

If a sequencer becomes unavailable, it is impossible to access read/write APIs that consumers are using and applications on the L2 network will be down for most users without interacting directly through the L1 optimistic rollup contracts. The L2 has not stopped, but it would be unfair to continue providing service on your applications when only a few users can use them.

To help your applications identify when the sequencer is unavailable, you can use a data feed that tracks the last known status of the sequencer at a given point in time. This helps you prevent mass liquidations by providing a grace period to allow customers to react to such an event.

Available networks

You can find proxy addresses for the L2 sequencer feeds at the following addresses:

- Arbitrum:* Arbitrum mainnet:0xFdB631F5EE196F0ed6FAa767959853A9F217697D
- Optimism:* Optimism mainnet:0x371EAD81c9102C9BF4874A9075FFFf170F2Ee389
- BASE:* BASE mainnet:0xBCF85224fc0756B9Fa45aA7892530B47e10b6433
- Metis:* Andromeda mainnet:0x58218ea7422255EBE94e56b504035a784b7AA204
- Scroll:* Scroll mainnet:0x95BcCbBBaCBB7ed224AC2EE38531f2467DF41Ea4

Arbitrum

The diagram below shows how these feeds update and how a consumer retrieves the status of the Arbitrum sequencer.

1. Chainlink nodes trigger an OCR round every 30s and update the sequencer status by calling thevalidatefunction in theArbitrumValidatorcontract by calling it through theValidatorProxycontract .
2. TheArbitrumValidatorchecks to see if the latest update is different from the previous update. If it detects a difference, it places a message in theArbitrum inbox contract .
3. The inbox contract sends the message to theArbitrumSequencerUptimeFeedcontract . The message calls theupdateStatusfunction in theArbitrumSequencerUptimeFeedcontract and updates the latest sequencer status to 0 if the sequencer is up and 1 if it is down. It also records the block timestamp to indicate when the message was sent from the L1 network.
4. A consumer contract on the L2 network can read these values from theArbitrumUptimeFeedProxycontract , which reads values from theArbitrumSequencerUptimeFeedcontract.

Handling Arbitrum outages

If the Arbitrum network becomes unavailable, theArbitrumValidatorcontract continues to send messages to the L2 network through the delayed inbox on L1. This message stays there until the sequencer is back up again. When the sequencer comes back online after downtime, it processes all transactions from the delayed inbox before it accepts new transactions. The message that signals when the sequencer is down will be processed before any new messages with transactions that require the sequencer to be operational.

Optimism, BASE, Metis, and Scroll

On Optimism, BASE, Metis, and Scroll, the sequencer's status is relayed from L1 to L2 where the consumer can retrieve it.

On the L1 network:

1. A network of node operators runs the external adapter to post the latest sequencer status to theAggregatorProxycontract and relays the status to theAggregatorcontract. TheAggregatorcontract calls thevalidatefunction in theOptimismValidatorcontract.
2. TheOptimismValidatorcontract calls thesendMessagefunction in theL1CrossDomainMessengercontract. This message contains instructions to call theupdateStatus(bool status, uint64 timestamp)function in the sequencer uptime feed deployed on the L2 network.
3. TheL1CrossDomainMessengercontract calls theenqueuefunction to enqueue a new message to theCanonicalTransactionChain.
4. TheSequencerprocesses the transaction enqueued in theCanonicalTransactionChaincontract to send it to the L2 contract.

On the L2 network:

1. TheSequencerposts the message to theL2CrossDomainMessengercontract.
2. TheL2CrossDomainMessengercontract relays the message to theOptimismSequencerUptimeFeedcontract.
3. The message relayed by theL2CrossDomainMessengercontains instructions to callupdateStatusin theOptimismSequencerUptimeFeedcontract.
4. Consumers can then read from theAggregatorProxycontract, which fetches the latest round data from theOptimismSequencerUptimeFeedcontract.

Handling outages on Optimism, BASE, Metis, and Scroll

If the sequencer is down, messages cannot be transmitted from L1 to L2 andno L2 transactions are executed. Instead, messages are enqueued in theCanonicalTransactionChainon L1 and only processed in the order they arrived later when the sequencer comes back up. As long as the message from the validator on L1 is already enqueued in theCTC, the flag on the sequencer uptime feed on L2 will be guaranteed to be flipped prior to any subsequent transactions. The transaction that flips the flag on the uptime feed will be executed before transactions that were enqueued after it. This is further explained in the diagrams below.

When the Sequencer is down, all L2 transactions sent from the L1 network wait in the pending queue.

1. Transaction 3contains Chainlink's transaction to set the status of the sequencer as being down on L2.
2. Transaction 4is a transaction made by a consumer that is dependent on the sequencer status.

After the sequencer comes back up, it moves all transactions in the pending queue to the processed queue.

1. Transactions are processed in the order they arrived soTransaction 3is processed beforeTransaction 4.
2. BecauseTransaction 3happens beforeTransaction 4,Transaction 4will read the status of the Sequencer as being down and responds accordingly.

Example code

This example code works on the Arbitrum, Optimism, and Metis networks. Create the consumer contract for sequencer uptime feeds similarly to the contracts that you use for otherChainlink Data Feeds . Configure the constructor using the following variables:

- Configure thesequencerUptimeFeedobject with thesequencer uptime feed proxy address for your L2 network.
- Configure thedataFeedobject with one of theData Feed proxy addresses that are available for your network.

// SPDX-License-Identifier: MITpragmasolidity^0.8.7;import{AggregatorV2V3Interface}from"@chainlink/contracts/src/v0.8/interfaces/AggregatorV2V3Interface.sol";/* **THIS IS AN EXAMPLE CONTRACT THAT USES HARDCODED VALUES FOR CLARITY. * THIS IS AN EXAMPLE CONTRACT THAT USES UN-AUDITED CODE. * DO NOT USE THIS CODE IN PRODUCTION.** */**contractDataConsumerWithSequencerCheck{AggregatorV2V3InterfaceinternaldataFeed;AggregatorV2V3InterfaceinternalsequencerUptimeFeed;uint256privateconstantGRACE_PERIOD_T**
* Network: Optimism mainnet * Data Feed: BTC/USD * Data Feed address: 0xD702DD976Fb76Ffc2D3963D037dfDae5b04E593 * Uptime Feed address: 0x371EAD81c9102C9BF4874A9075FFFf170F2Ee389 * For a list of available Sequencer Uptime Feed proxy addresses, see: * https://docs.chain.link/docs/data-feeds/l2-sequencer-feeds */constructor(){dataFeed=AggregatorV2V3Interface(0xD702DD976Fb76Ffc2D3963D037dfDae5b04E593);sequencerUptimeFeed=AggregatorV2V3Interface(0x371EAD81c9102C9BF4874A9075FFFf170F2Ee389);}// Check the sequencer status and return the latest datafunctiongetChainlinkDataFeedLatestAnswer()publicviewreturns(int){// prettier-ignore(/*/(uint80 roundID/*/,int256answer,uint256startedAt,/*/(uint256 updatedAt/*/,uint80 answeredInRound/*/)=sequencerUptimeFeed.latestRoundData();// Answer == 0: Sequencer is up// Answer == 1: Sequencer is downboolisSequencerUp=answer==0;if(!isSequencerUp){revertSequencerDown();}// Make sure the grace period has passed after the// sequencer is back up.uint256timeSinceUp=block.timestamp-startedAt;if(timeSinceUp<=GRACE_PERIOD_TIME){revertGracePeriodNotOver();}// prettier-ignore(/*/(uint80 roundID/*/,intdata,/*/(uint startedAt/*/,/*/(uint timeStamp/*/,/*/(uint80 answeredInRound*/)=dataFeed.latestRoundData();returndata;}} Open in Remix What is Remix? ThesequencerUptimeFeedobject returns the following values:

- answer: A variable with a value of either1or0* 0: The sequencer is up
- 1: The sequencer is down
- startedAt: This timestamp indicates when the sequencer changed status. This timestamp returns0if a round is invalid. When the sequencer comes back up after an outage, wait for theGRACE_PERIOD_TIMEto pass before accepting answers from the data feed. SubtractstartedAtfromblock.timestampand revert the request if the result is less than theGRACE_PERIOD_TIME.

If the sequencer is up and theGRACE_PERIOD_TIMEhas passed, the function retrieves the latest answer from the data feed using thedataFeedobject.