Block propagation is the mechanism used to transmit state information (blocks) to all the nodes in a blockchain, including L2 blockchains or rollups. Block propagation is key for blockchain nodes to receive any new released block in time, in order to maintain its local view of the blockchain state updated. Deriving from the decentralized blockchain architecture, block propagation usually rely on a decentralized P2P network, where nodes have to discover and connect to other blockchain nodes in order to update its network state and propagate it to the rest of the network. Block propagation faces some challenges, such as scalability, performance, efficiency and security. In this post we analyze these challenges and we evaluate the potential solutions for a block propagation solution for Dymension rollups [1].

Related work

In the following table we list the main optimistic rollups and its solution for block propagation. We analyse optimistic rollups because are the closest to Dymension in terms of performance, challenges and objectives:

L2 Network

Block time

Block propagation approach

Specific parameters

Optimism

2 seconds

Gossipsub [2]

Gossipsub defaults https://github.com/ethereum-optimism/optimism/blob/65ec61dde94ffa93342728d324fecf474d228e1f/op-node/p2p/gossip.go#L38

Arbitrum

0.25 seconds

Connected to a list of feeds given by the configuration (centralized solution). [3]

By default all nodes are connected to arbitrum feed wss://arb1-feed.arbitrum.io/feed

opBNB

1 second

Gossipsub [2]

Use same optimism parameters https://github.com/bnb-chain/opbnb/blob/afa3cac33a2543b47524aaf2c2b60e237d48d02c/op-node/p2p/gossip.go#L38

Polygon PoS

2 seconds

Eth protocol [4]

Original geth parameters

We can observe that most of the main existing optimistic rollups rely either on gossipsub protocol or eth protocol; both are gossip-based protocols (explained in the next section) and are the most common solutions in blockchain for block propagation. However, arbitrum is an exception. It utilizes a sort of centralized solution, where all nodes connect by default to the same arbitrum block feeder (which is probably directly connected to the sequencer). Any node can select a different source, but by default the configuration of any arbitrum node points to the same address. This solution is probably derived from the fact that is the rollup with smaller blocktime and they want to keep the best performance, sacrificing other aspects such as decentralization.

Others block propagation solutions

Other interesting block propagation solution are solutions based on Erasure Coding. Erasure coding extends a block that is split in M chunks, to M+N=K, where N are redundancy chunks. Any node that receives any M messages out of the total K messages can recover the whole message and get the block. Erasure coding can help in providing reliability, without having to send multiple copies of the same block, as Gossipsub does, saving network traffic and being more efficient. However it may require more complex dissemination and it is not as proven and as stable as Gossipsub, but it is worth to explore for

future research directions and future releases.

Examples:

- Solana: Turbine (erasure coding) [5]
- Harmony [6]

Therefore, we select Gossipsub as the main candidate for Dymension rollapps block propagation mechanism.

GossipSub approach

Gossipsub is the mechanism used by Ethereum consensus layer and Filecoin [2] network for block propagation. Gossipsub is a mechanism that tries to build a dissemination tree for a specific communication group (called channel or topic) with fixed number of links between nodes in order to be scalable. On top of the message dissemination to direct connected nodes, nodes send a list of messages received (without sending the message) to other nodes in order to provider higher reliability in case of faulty nodes and churn to avoid network partitions. In case a node receives an advertisement for a message that did not receive yet, it can request the message on demand. Gossipsub specs details are the following:

- Gossipsub nodes try to establish a fixed number of links D with other nodes that are

part of the same propagation group (i.e., pubsub channel or topic).

- Every period t (heartbeat parameter), nodes check about their links to other peers:

◦ If the number of links with other peers in a specific node is below D_low

threshold, the node tries to establish new links till reaching D.

◦ If the number of links with other peers in a specific node is above D_high

threshold, the node drop links till reaching D.

- Every message received by a node, is resent to all the directly connected peers, in exception of the link that the message is received from, or in case the source of the message is one of the directly connected nodes.
- Nodes cache every message received and keep them during a short period (this period should be big enough to cover the period between batches written to Data Availability (DA) module).
- Every period t, every node sends and advertisement (not the messages) of the list of cached messages to D random nodes different from the ones directly connected . In case a node receives an advertisement for a message that does not received yet, it can request it.
- Nodes creates a peer score, in order to select the best peers to connect. This

peering score can include parameters such as delivery rate, time connected or IP

addresses, and can also help reducing the effect of sybil nodes.

- Gossipsub relies on an external peer discovery that is not part of the mechanism (I.e, it needs to know who is part of the propagation group in order to find nodes in the same group and propagate blocks efficiently). It usually uses Kademlia DHT in order to discover any node in the network, performing lookups to random nodes in the network.

Gossipsub is aligned with the main objectives of the block propagation, listed below:

- Efficiency: The overhead on the sequencer is fixed and is given by the number of links D. The sequencer is the only node who publishes data to the group, and will only need to send D copies of the block and it will be propagated to the all the nodes in the network reliably. However, this overhead is not negligible (D times the block in average) and there may be more efficient solutions.
- Fault tolerance: Gossipsub is highly tolerant to faulty nodes. From one side, new

links are established in case nodes go down. In addition to this, nodes send gossip

(list of all messages) to avoid network partitions and to ensure all nodes receive the

messages.

- Scalability: Gossipsub is highly scalable. The amount of links each node has to

keep, and therefore the number of messages copies sent by each node, is fixed and is not growing by the number of nodes

in the network. Obviously, more nodes

in the network will require more propagation hops, however the number of nodes

reached after each hop is exponential and with only 6 hops, with a number of links

D=6, it can reach up to 20k nodes.

- Latency: The latency will be higher compared to all nodes receiving a copy directly

from the sequencer, but this is not scalable. However, the latency will keep bounded

and will be given mainly by the number of hops required to reach each node, which

is low, and the bandwidth capacity of the nodes in the network.

- Security: The main attack vectors for a block propagation could be the following:

- Spam attack: Any node could inject fake traffic, wasting network resources and avoiding the correct reception of valid blocks. In this case we can assume the sequencer is known by all nodes. Therefore this could be easily avoided by checking the signature of the block and if it does not correspond to the sequencer, do not propagate the block. This will add some delay for the validation of each block, but it is necessary to eliminate spam attacks. In addition to this, a safety check would be to validate the block locally before sending it further.

- Sybil nodes / Network partition attack: In a permissionless network it is very

easy to generate nodes controlled by the same entity to attack the network.

Sybil nodes can be added to the network and act maliciously. In a rollup scenario , the

attack could be just avoiding the correct transmission of blocks. This can lead to

network partitions and nodes not receiving the blocks. In order to avoid that, the

peer scoring function included in Gossipsub v.1.1 can help. This function can limit the number of sybil nodes a node is connected to and avoid being eclipsed by them (having all connections to only sybil nodes). The function can include parameters such as IP addresses (it is very easy to generate sybils from the same machine, but having sybils

placed in different networks it require some cost), messages sent history (avoiding to connect to nodes that does not propagate blocks correctly can help) , or time connected (sybil nodes tend to be less stable). Also id distribution can help, since sybil nodes tend to have an id distribution that is not uniform.

- Spam attack: Any node could inject fake traffic, wasting network resources and avoiding the correct reception of valid blocks. In this case we can assume the sequencer is known by all nodes. Therefore this could be easily avoided by checking the signature of the block and if it does not correspond to the sequencer, do not propagate the block. This will add some delay for the validation of each block, but it is necessary to eliminate spam attacks. In addition to this, a safety check would be to validate the block locally before sending it further.

- Sybil nodes / Network partition attack: In a permissionless network it is very

easy to generate nodes controlled by the same entity to attack the network.

Sybil nodes can be added to the network and act maliciously. In a rollup scenario , the

attack could be just avoiding the correct transmission of blocks. This can lead to

network partitions and nodes not receiving the blocks. In order to avoid that, the

peer scoring function included in Gossipsub v.1.1 can help. This function can limit the number of sybil nodes a node is connected to and avoid being eclipsed by them (having all connections to only sybil nodes). The function can include parameters such as IP addresses (it is very easy to generate sybils from the same machine, but having sybils

placed in different networks it require some cost), messages sent history (avoiding to connect to nodes that does not propagate blocks correctly can help) , or time connected (sybil nodes tend to be less stable). Also id distribution can help, since sybil nodes tend to have an id distribution that is not uniform.

Gossipsub vs Eth protocol

Eth protocol its very similar in concept to Gossipsub but with different parameters. The main difference is that Gossipsub is a generic pubsub protocol that can work with any protocol, and eth is a specific protocol that was built specific for Eth PoW network.

This approach was used in the Ethereum execution layer (proof-of-work), but after the PoW-to-PoS transition (The Merge), block propagation is no longer handled by the this protocol, and is only handled by consensus layer, which is actually using a version of Gossipsub protocol.

Geth protocol and Gossipsub are very similar and they rely on the same design rationale, which is that all nodes connect to a random subset of nodes in the network and when they receive a block from one peer they propagate it to the rest of the peers they are directly connected to. However Gossipsub is a more generic protocol that it allows multiple broadcast channels, while Eth protocol is a custom implementation protocol for Ethereum block propagation.

In general, Gossipsub protocol should be able to cover all the challenges. Therefore we believe Gossipsub is the best candidate for a decentralized rapid block propagation for Dymension rollups. However it can never assure a specific performance (max block propagation time) and it will depend on the size of the network (i.e, the number of hops it needs to propagate a new block to all peers) and the quality of the links between peers (latency and bandwidth).

To this end, in the following section we evaluated Gossipsub protocol performance using a network simulator, in order to assess the correct performance in realistic scenarios and to understand what are the best parameters to use in a rollup for a correct block propagation under specific network conditions (e.g. number of nodes with specific bandwitdh/latency requirements)

Evaluation

For the performance evaluation we focused on the block propagation time, which is the most important metric to evaluate how nodes can be updated as fast as possible. We also include other metrics related to efficiency, such as number of replicated blocks received or hops required to reach the destination, and recovery in case nodes go down.

We analysed the performance of the solution simulating the approach in a network

simulator called Peersim [7], a scalable P2P network simulator. We extended the functionality of Peersim, adding an implementation of Gosispsub protocol (version 1.0), available in this github repository [8].

In the evaluation we used the following parameters:

- D=8, D_high=12, D_low=4.
- Default network size: 50 nodes
- Latency between nodes: 5-100 ms
- Node bandwidth: 100Mbps.
- Hearbeat period t: 1 second
- New block every 0.2 seconds.
- Simulation time: 250 blocks
- Block size: 100KB.

Node size

We evaluated the block propagation time for three different network sizes: 10 nodes, 50 nodes and 100 nodes, to observe how evolves the performance of the network when increasing network size and to see if it is possible to scale using this approach. In this part of the evaluation there is no churn. The figure represents the Cumulative Distributed Function (CDF) of the average time to receive a block by each node in the network.

Figure 1: Block propagation time evaluating different network size

We observe the block propagation time (Figure 1) (time between the sequencer releases a block and any node receives it) for 10, 50 and 100 network size. We can see the block propagation time is below 200ms in most cases and increasing the network size slightly increases the block propagation time, but without having an important degradation of the network performance and allowing a scalability of the rollup.

Figure 2: Average block propagation time of all nodes per block

In Figure 2 we show the same metric but observing the evolution of the average propagation time during 250 blocks. We can see there is no big difference between different blocks and the performance keeps stable over time.

Block size

Figure 3: Block propagation time for different block size

In Figure 3 we analysed block propagation time for different block size, in a 50 nodes network. We can observe that when we increase the block size the block propagation time also increases. From 100KB to 200KB the block propagation time does not seem to be substantially affected. However, when we increase the block size to 300KB we clearly see a performance degradation caused by network congestion.

Therefore we select a 100KB as maximum block size

in order to avoid congesting nodes when sending too much data.

Link degree (D)

Figure 4: Block propagation time using different node degree (D)

Link degree -D- (number of links that each node tries to establish for a dissemination group at the overlay level) is a configurable parameter in Gossipsub. We observe that, in a network size of 50 nodes, there is no big impact in increasing the number of connections -degree(D)- in the block propagation performance. Actually the performance is slightly better with a degree D of 8. It maybe caused by the fact that with smaller degree there are less copies sent and the bandwidth required is smaller. Therefore, we select 8 as default node degree D

in order to avoid sending unnecessary message replications to the P2P network. Since increasing the link degree increases the reliability but also number of replicates in the network and therefore increases the network utilisation.

Number of hops

In Figure 5 we observe the number of hops a block must do to reach all nodes. We observe for smaller networks the number of hops is very small (max 3) but for bigger networks (50-100) this value is higher. However, the number of hops grows logarithmic, being equal to 5 in the worst case.

Overhead

Figure 6: Times the same block is received by a node

In Figure 6 we observe the times the same block is received by a node using Gossipsub, which is one of the main drawbacks that limits its propagation capacity. We can see the number of replicates received is in the order of the degree used, since any node can received a copy from any node it is directly connected to.

Recovery

Figure 7: Block propagation time for a node added to the network at block 45

In Figure 7 we observe the performance of block propagation time when nodes are leaving and joining the network. We see the block propagation for a specific node that joints the network at block 45. We observe, that even joining the network at block 45, is able to get all previous blocks (in this case nodes need to cache this previous 45 blocks) but with a latency higher than 1 second . This is caused because of the time it needs to request previous blocks from other nodes, and is triggered after each heartbeat period, which is 1 second. We also see other intervals with spikes in the propagation times. This is caused by other nodes leaving or joining, however after a short time the performance comes back to the optimal rate.

Conclusions

In this document we analysed different solutions for block propagation in rollups, and we analysed the performance of Gossipsub protocol in a network simulator. We concluded the best block parameters for Dymension rollapps are using a maximum block size of 100KB each 0.2 seconds and a link degree D equal to 8 for the Gossipsub protocol.

References

[1] Dymension Rollaps https://docs.dymension.xyz/learn/rollapps/dymint

[2] GossipSub: Attack-Resilient Message Propagation in the Filecoin and ETH2.0 Networkshttps://arxiv.org/abs/2007.02754

[3] Arbirtrum docs. https://docs.arbitrum.io/node-running/how-tos/read-sequencer-feed

[4] Ethereum Wire Protocol https://github.com/ethereum/devp2p/blob/master/caps/eth.md

https://medium.com/harmony-one/block-syncing-in-1-36s-with-harmonys-adaptive-ida-protocol-de5da398569e

[5] Turbine. Solana’s block propagation Protocol Solves the Scalability Trilemma https://medium.com/solana-labs/turbine-solanas-block-propagation-protocol-solves-the-scalability-trilemma-2ddba46a51db

[6] Harmony technical whitepaper https://harmony.one/whitepaper.pdf

[7] PeerSim P2P Simulator https://peersim.sourceforge.net/

[8] P2P simulator for Kademlia and Gossipsub GitHub - srene/p2p-simulator: A Kademlia simulator based on PeerSim