Thanks to TXRX research team, @AgeManning, @protolambda, @djrtwo, and @benjaminion for the support provided

The purpose of this post is to present a view of the gossip data that I collected while monitoring the Witti testnet. Similar to flight test analysis in the aerospace world, testnet data can be used to validate assumptions and flag unexpected behavior. This is just my first attempt at trying to make sense of what I am seeing. The more eyes on this the better so comments/feedback/suggestions are welcome.

Configuration

The data for the following analysis was collected by a single instance of theimp network agent.

- Testnet:

Witti

- Collection Dates:

June 10, 2020 - June 13, 2020

The network agent uses Sigma Prime's implementation of gossipsub that they contributed torust-libp2p. Minor modifications were made to gossipsub params:

- mesh_n_high:

set to the estimated number of validating nodes

- mesh_n_low:

set to 1/3 of the estimated number of validating nodes

- mesh_n:

set to 2/3 of the estimated number of validating nodes

- gossip_lazy:

set to 0

In addition, the gossipsub Least Recently Used (LRU) cache was remove to enable the logging of duplicate messages.

Analysis

To start, let's take a peek at some summary level statistics of the data collected.

- Starting slot:

105787

- Ending slot:

129697

- Number of slots:

23910

- Number of blocks received:

268089

- Average number of blocks received per slot:

11.2

- Mean block size:

1909

- Median block size:

1584

- Number of peers:

19

- Number of peers validating:

12

- Number of peers not validating:

7

Question: 19 peers and 11 additional received gossip blocks seems a bit excessive, right? It would be interesting to see the number of duplicate messages using normal mainnet params.

Next, let’s take a look at the number of times each validator was selected as a block proposer.

[

993×351 25.2 KB

](https://ethresear.ch/uploads/default/original/2X/f/f0f177b340dc6d3299b9a6cfaf7bba8dd2c3075b.png)

This plot was generated provide a bird’s eye view of the number of times each proposer is selected. The x-axis isn’t labeled, but each line represents the number of times a particular proposer index was selected (according to unique blocks received).

Notice how some proposer indexes appearing so rarely in blocks received? This can happen due to a combination of skipped slots and low balances (which will affects the probability of selection).

Let’s zoom in for a closer look.

[

993×539 45.4 KB

](https://ethresear.ch/uploads/default/original/2X/b/b1b52cd5318436fa6fb5ce96f64ad4d013bd0eda.png)

This plot is the same as the previous, but the x-axis is zoomed in to show the actual proposer indexes in question.

Now that we are here…there seem to be a few missing validator indexes. It’s worth counting the missing blocks/possible skipped slots:

block_slots_received = set(df['slot'].to_list()) slots=set(range(df['slot'].min(),df['slot'].max())) print("Number of missing block slots:", len(slots.difference(block_slots_received)), "of", df['slot'].max()-df['slot'].min())

Number of missing block slots: 5319 of 23910 (22%)

Question: Are there that many skipped slots on the Witti testnet, or are clients frequently having to request missing blocks from peers?

Next, let’s take a look at arrival times of the first block in a slot.

[

993×557 102 KB

](https://ethresear.ch/uploads/default/original/2X/a/a4c73a72b3fbe59a4dc51ec27aecb94e3617a768.png)

This plot zooms in on the y-axis to look at the earliest arriving blocks. No need to adjust your picture…some blocks seem to be arriving before the slot has started. This isn’t unexpected because node clocks will never be perfectly synchronized.

Question: The networking spec tells clients to filter messages arriving more than 500 milliseconds early. Any thoughts on how we should validate this number?

Now’s probably a good time to mention that the slot is taken from the slot as specified in the block. Arrival times are computed relative to slot times which are calculated as follows:

arrival time = timestamp - block.slot*12.0 - GENESIS_TIME

It might be interesting to get a feel for arrival time variance per block. If you remember, I disabled the LRU cache in the gossipsub implementation to ensure I could easily log duplicate blocks.

[

993×563 89.7 KB

](https://ethresear.ch/uploads/default/original/2X/7/78a9fdd5d853446f11166246d47ea28a3c08ed9c.png)

This plot gives a view of the arrival time variance of blocks for each slot. The obvious feature of this plot is the unexpectedly large arrival time variances. Let's make sure this isn't due to a single peer (or client type).

[

594×530 43 KB

](https://ethresear.ch/uploads/default/original/2X/1/1cd4a85a35e3090edd807992f6aa9c4d4e978637.png)

This chart shows us that every peer is (at some point) responsible for propagating a block a couple hundred seconds late. This doesn't seem

to be an implementation specific issue.

Note: It's not ideal that duplicate blocks are still arriving several minutes after the initial block. Perhaps we should consider more agressive measures to prevent propagation of old blocks.

Let's zoom in for a closer look at these arrival time variances.

[

993×568 375 KB

](https://ethresear.ch/uploads/default/original/2X/9/9e62065dd3f2171858fde386efcd86c7610ec833.png)

This plot is the same as the previous, but we have zoomed in on the y-axis. Darker colors represent smaller blocks and lighter colors represent larger blocks. The closer the dot is to the x-axis, the smaller the variance in arrival times. Notice the strata in colors? Smaller blocks (darker dots) seem to have less arrival time variance (closer to x-axis) than larger blocks (lighter dots).

Let's see if the relationship holds if we group by message size.

[

998×759 111 KB

](https://ethresear.ch/uploads/default/original/2X/7/751ff0c6d4bbdb1b8f2465e254748023a0e5e3e7.png)

This plot tells a us few things. Let's start with the upper plot. There are two marker colors on the plot, red and green:

Including Duplicate Blocks (Red Marker)

Shows us the trend in mean block arrival times (red dots) and arrival time std dev (red x) as block size increases.

First Block Arrival (Green Marker)

Shows us the trend in mean first

block arrival times (green dots) and first

block arrival time std dev (green x) as block size increases.

The bottom plot represents the block count by size. I should also mention that arrival times > 12 seconds were filtered out before calculating mean and std dev.

There does seem to be a relationship between message size and arrival variance when accounting for duplicate block messages in the population; however, this relationship is less pronounced when only considering the first block message received.

Question: Why would message size affect arrival time variance? Perhaps block validation affects arrival times. The more hops away the sending peer is from the block proposer, then more times it has been validated before propagating - this could add up. If larger blocks take more time to validate (plus a little longer to send), then this could explain the greater variance observed with larger messages. I would like to collect more data on this before drawing any real conclusions.

Questions

- 19 peers and 11 additional received gossip blocks seems a bit excessive. It would be interesting to see the number of duplicate messages using normal mainnet params. Has anyone else looked at duplicate blocks received on an eth2 testnet? With normal mainnet params, how many duplicates do we consider acceptable?

- The spec mentions that MAXIMUM_GOSSIP_CLOCK_DISPARITY

is set to 500 milliseconds as a placeholder and will be tuned based on testnet data. How do we want to validate this number?

- The network agent didn't receive blocks for 22% of the slots. I'm curious if there are that many skipped slots on the Witti testnet, or are clients frequently having to request missing blocks from peers?

Suggestions

Propagation of Old Blocks

Blocks that are several hundred seconds old are still being propagated on the network by peers. This isn't isolated to a single peer; all of the agent's peers are guilty of it - even ones that are successfully hosting a large number of validators. This tells me that when the occasional block arrives late, these peers have already requested it via rpc; however, they choose to forward the block bc they haven't received it (or don't remember receiving it) via gossip. I don't see how propagating a block that is half a dozen (or more) slots behind is helping consensus? I suggest:

- tightening the bounds defined in the spec for propagating old blocks.

If clients are relying on the memcache in GossipSub to check for messages they have already seen, then it is possible that the history_length parameter is part of the problem. If this is the case, perhaps we should consider:

- updating the history_length

GossipSub param from 5 seconds to at least 12 seconds to match SECONDS_PER_SLOT

.