**After-Action Report of 2019-08-23 Factom Network Pause**

One of the features of the Factom Protocol is that it includes a mechanism for a leader to be upgraded on the fly avoiding any hiccups in process during upgrades. We call this mechanism brain swapping. The short version is a second machine is spun up as a follower with the upgraded software and once the new machine is in sync with the network the ANO picks a future block height when the new machine will assume the identity of the leader and the leader assume the identity of the follower. So until block N machine 1 is the leader for the ANO and after block N machine 2 is the leader for the ANO.

The initial pause was caused by a failed brainswap. Both the leader and his intended replacement acted as leader providing signatures (DBSigs messages and ACK messages) for block 206920. This paused the network as both were validly signed and a majority of the leader choose one DBSig while a majority of the Audits chose the other DBSig. As a result no sequence of elections could compose a full set of leader in consensus. Without reaching consensus the protocol will not go forward as that would allow conflicting data to be recorded.

The network had stopped progressing forward at 0430 UTC 2019-8-23 (11:30 PM Texas time).

The first restart attempt exposed several federated servers as having corrupted balances, which resulted in a panic claiming “Too few factoids”. This was also a self protection mechanism guarding against negative balances on the network. The first attempted fix was for ANOs to delete the FastBoot file, which contains the corrupted state that was causing the panic. Rebuilding the savestate files is an operation which takes over an hour, because factomd needs to rescan the blockchain from the beginning. There were also several Leader nodes which did not save the highest block 206919. They still had block 206918 in their database. If they started up with the rest of the network, they would not be able to participate in consensus. In order to facilitate more reliable starting of the network, ANOs who had nodes at block height 206918 moved their keys to backup nodes which were on 206919. This was initiated around 0630 UTC 2019-8-23 (1:30 AM Texas time).

Both of these operations, moving identities and rebuilding save, were complete to a substantial point around the network at around 0900 UTC 2019-8-23 (4 AM texas time) a network restart was attempted. Consensus was not achieved as several of the nodes still suffered from the “Too few factoid” panics even after rebuilding savestates.

There was a pending fix for this bug that was contained in the next planned factomd release version (Parchment, v6.3.3). As a quick response, ANOs updated to the Parchment release in the early morning hours. This process also required rebuilding savestates. The call to upgrade to 6.3.3 went out around 4:30 AM Texas time. This took several hours for ANOs to upgrade and for their leaders to get into a state where they could potentially participate in a reboot. This began around A network reboot was attempted 2 times over the next 6 hours. ANOs had updated by 7:30 AM to begin the restart attempts. The network did not successfully build a block using the Parchment release.

A new version of factomd v6.4.1 [Cotton] (was prepared after the failed restart attempts. The new version of factomd was based on v6.4.0 [Origami] (which was based on v6.3.2 [Bond]) but had a few fixes in it.

* A backport of the savestate issue causing the “Too few factoids” panic
* A change allowing savestates (named FastBoot\_MAIN\_v12.db) built with Parchment to be allowable when booting Cotton.
* Changing the message filter time to allow messages created 20 minutes before boot time. This would allow nodes that were booted slightly after another one to accept their peer’s DBsigs. DBsigs are sent immediately at boot, but peer’s messages that are generated before a local boot time are ignored. The assumption was that nodes when booted were filtering out Directory Block Signatures from leaders booted seconds earlier.

<https://github.com/FactomProject/factomd/compare/v6.4.0...v6.4.1>

While Cotton was being deployed and tested by the Core Committee, a new bug was identified. Under some circumstances, when booting v6.4.1 a savestate file was not created. It resulted in savestate files not being saved to disk, which would make leaders not able to participate in consensus on boot. Luckily, most of the ANOs had previously rebuilt their savestates with Parchment, which created usable savestate files. The bug was visible in logs as:

**SaveState.MarshalBinary err:BinaryMarshallable is nil...**

This error was attempted to be debugged and general deployment was delayed by an hour and a half while looking for this bug. Not all nodes exhibited this behaivior. After that amount of time passed without finding the bug, a chance was taken to have a general deployment and rely on the Parchment savestates to bring up a majority of the nodes. Getting the network running was worth the chance.

In simulation the bug was found to be reproducible. The bug was fixed, but did not make it into the Cotton release deployed as 6.4.1.

<https://github.com/FactomProject/factomd/commit/aab2a4bab8168e58b67015ade0e20030dd686745#diff-838b45ab9ee95cc956451226977fdd64R669>

At 5 PM Texas time the network restart was attempted again. It did not succeed within an hour and a half.

On the final reboot attempt the root cause was discovered to be related to network followers failing to propagate messages, detailed below.

We have come to understand the issue that inhibited the first few attempts to restart of the network. The original issue was further complicated after being paused for more than an hour. The followers’ MessageTimeFilter configuration caused followers to stop gossiping messages from leaders. The backhaul network (which is intended to ensure leader to leader communication is efficient) proved inefficient once the follower network stopped gossiping.

Here is the more technical explanation.

The Factom gossip protocol works by having a node randomly choose 16 of their neighbors to send a copy of each valid message as it arrives. The pool of neighbors for a given node is typically 60 to 80 peer nodes. In theory every node will get on average 16 copies of every message. Because the 16 peer nodes are selected for broadcast at random, each node may receive from 0 to more than 16 copies of the same message. Because of this design, there is a missing message protocol that allows a node to ask a neighbor for a copy of a message when a given node is detects that it is missing a message. The missing message protocol works by asking a random neighbor for the missing message, and will try a different neighbor if missing message is not provided within a small timeout period.

The backhaul network is a set of nodes that is primarily intended to ensure that every leader is no more than two hops away from every other leader. It serves three secondary purposes:

1. Improves protocol performance
2. Allow the leader to communicate even if there is an issue with the follower network … (in theory).
3. The backhaul peers serve as seed peers that provide the first connections when a new follower first joins the network.

The backhaul is currently implemented as four follower nodes who are “special” to all the leaders. Special nodes always receive a copy of every outbound message, so these four nodes are always guaranteed to be one hop away from every leader and to have every message.

The Factom Protocol’s consensus algorithm is shielded from duplicate messages inherent in the gossip protocol with a filter that remembers all messages within a two hour window. Normally this time window is limited to one hour prior and one hour in the future. Obviously the majority of the future part of the filter window remains unused as long as a node’s local clock is close to the actual time.

**What went wrong**

A known bug that could result in incorrect balances when restoring from savestate on boot was still in the deployed code base. On the first reboot after the pause this bug caused a significant number of authority nodes to have a balance error for an account that had a transaction in the last block loaded and, to prevent the perceived double spend, these servers panicked on boot. This required the ANOs to upgrade to a newer code base (6.3.3 then 6.4.1). This took more than an hour. When the next reboot occurred the follower network (which cannot be automatically rebooted) had a message filter timestamp that matched the start of the previous block which was now three hours in the past, so when new messages arrived to follower (who had not been rebooted) the message appeared to be invalid (because it came from more than one hour in the future) and so was not gossiped.

Had the backhaul network worked correctly, the leaders would have moved on and completed the next block. The followers would have received the block as a DBSTATE and reset their message filters to the time of the latest block and started gossiping again. The four nodes of the backhaul received every message from the leaders and correctly transmitted them to 16 random peers. However, since the 29 leaders were not "special" to the backhaul nodes, they were just a small fraction of the 180 neighbors and therefore did not directly receive all the messages. The messages were not indirectly received by the leaders either because the followers that should have passed them along dropped these messages due to the time filter. Note that higher number neighbors (180) is the result of the backhaul nodes also being the peer seed nodes.

Because the leaders and backhaul nodes had been rebooted their message time filter was reset to boot time, so they would still gossip and accept the messages but the leaders had only a 29 out of 180 chance being sent the message. To further complicate the matter, leaders also allow follower connections, which typically had 80 connections, so when they asked for the missing messages using the missing message protocol their chances of asking a backhaul node was 4 out of 80. So they typically had to ask 20 times to get the message they needed. The timeout between times to ask is 30 seconds. The timeout for elections to begin on a particular node is 2 minutes. If a leader node was not able to get a particular message within the 2 minutes they would start an election even though no election was needed.

In the several reboot attempts the network struggled through several minutes and many elections but with the severely reduced gossip network the reboot attempts were unable to complete the next block. Once the issue was understood more followers were rebooted along with the rest of the Leaders. This was enough to get the network bast a single block. Once the block completed it was sent by DBSTATE to the adjacent non-rebooted followers which updated their message filter timestamps and then they sent messages on to other followers and over the next few minutes the follower network resumed gossiping and network operations returned to normal. The first block was finalized around 0000 UTC 2019-8-24 (7:00 PM texas time)

The network was back online and progressing at around 0130 UTC 2019-8-24 (8:30 PM texas time). The network had stopped forward progress for about 19.5 hours.

Mitigations to minimize issues uncovered in this pause:

* Increase the chattiness of the backhaul network to ensure more messages are passed to leaders when other followers are malfunctioning.
* Change this message filter to respond and reset to more then just forward movement of the blockchain to also reset during more common ANO things. For example, hard messages which would happen when the servers are started, but before the blockchain is moving forward.
* Reduce the period when asking for missing messages, so that nodes can catch up quicker when a lot of the network is missing the things they’re asking for, so they can more rapidly find the thing they’re missing.
* Fix the savestate race condition.