Signage and Infusion Points

Each sub-slot in both the challenge chain and the reward chain is divided into 64 smaller VDFs. Between each of these smaller VDFs is a point called a **signage point**. Timelords publish the VDF output and proof when they reach each signage point.

> The challenge and reward chains both have signage points. The infused challenge chain, however, does not.

The signage points occur every 9.375 seconds (64 signage points per 600-second sub-slot). The number of iterations between each signage point is **sp_interval_iterations**, which is equal to sub-slot_iterations / 64.

The challenge at the start of the sub-slot is also a valid signage point. As each of the 64 signage points in the sub-slot is reached, those points are broadcast, starting from the fastest timelord's beacon client, and propagating to every other beacon client on the network.

Farmers receive these signage points and compute a hash for each plot, at each signage point. If the hash starts with nine zeros, the plot passes the filter for that signage point, and can proceed. This disqualifies around 511/512 of all plot files in the network, for that signage point. The formula to compute the filter hash is:

```
plot filter bits = sha256(plot id + sub slot challenge + cc signage point)
```

The proof of space challenge is computed as the hash of the plot filter bits:

```
PoSpace challenge = sha256(plot filter bits)
```

Using this challenge, the farmers fetch quality strings for each plot that made it past the filter. Recall that this process requires around seven random disk seeks, which takes around 70 ms on a slow HDD. The quality string is a hash derived from part of the proof of space (but the whole proof of space has yet to be retrieved).

> For both of our previous example, as well as the next example, we'll use the following values:
> sub_slot_iterations = 100,000,000
> sp_interval_iterations = sub_slot_iterations / 64 = 1,562,500

The farmer computes the **required_iterations** for each proof of space. If the required_iterations < sp_interval_iterations, the proof of space is eligible for inclusion into the blockchain. At this point, the farmer fetches the entire proof of space from disk (which requires 64 disk seeks, or 640 ms on a slow HDD), creates an unfinished beacon block, and broadcasts it to the network.

> For the vast majority of eligible plots, required_iterations will be far too high, since on average 32 will qualify for the whole network for each 10-minute sub-slot. This is a random process so it's possible (though unlikely) for a large number of proofs to qualify. The signage_point_iterations is the number of iterations from the start of the sub-slot to the signage point. Any plot that does meet the required_iterations for a signage point will qualify as there is no rivalry between winning plots.

The exact method for required_iterations is the following:

```
sp_quality_string = sha256(quality_string + cc_signage_point)
required_iterations = (difficulty
    * difficulty_constant_factor
    * int.from_bytes(sp_quality_string, "big", signed=False)
    // pow(2, 256) * expected_plot_size(size))
```

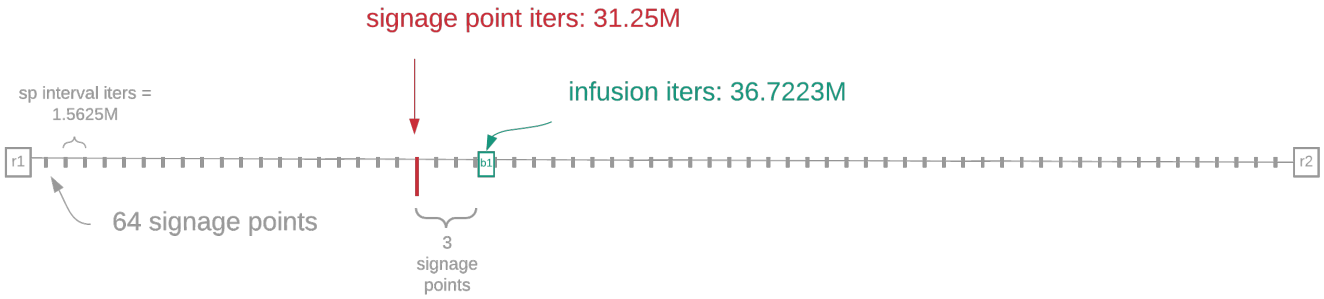
The difficulty constant factor is based on the initial constants of the beacon chain. For BPX, it is `2^64`. The difficulty varies per epoch. As you can see, the **sp_quality_string** is converted into a random number between 0 and 1, by dividing it by `2^256`, and then multiplied by the plot size.

For consensus purposes, the `expected_plot_size` is `((2 * k) + 1) * (2 ** (k - 1)).`, where k>=32<50. The actual plot size is that value times a constant factor, in bytes. This is because each entry in the plot is around `k+0.5` bits, and there are around `2^(k)` entries.

The **infusion_iterations** is the number of iterations from the start of the sub-slot at which the block with at least the required quality can be included into the blockchain. This is calculated as:

`infusion_iterations = ( signage_point_iterations + 3 * sp_interval_iterations + required_iterations) % sub-slot_iterations`

Therefore, infusion_iterations will be between 3 and 4 signage points after the current signage point. Farmers must submit their proofs and blocks before the infusion point is reached. The modulus is there to allow overflows into the next sub-slot, if the signage point is near the end of the sub-slot.



A drawing shows the infusion point as a green square marked `b1`. The first and last blocks of the sub-slot are marked `r1` and `r2`, respectively. For this example, the farmer will create the block at

the time of the signage point marked with a red arrow, which we'll call `b1'`.

At `b1`, the farmer's block gets combined with the VDF output for that point. This creates a new input for the VDF from that point on, i.e. we infuse the farmer's block into the VDF. `b1` is only fully valid after two events have occurred:

1. infusion_iterations has been reached, and
2. Two VDF proofs have been included: one from `r1` to the signage point and one from `r1` to `b1`. (Actually it's more since there are three VDF chains).

The farmer creates the block at the time of the signage point, `b1'`. However, `b1'` is not finished yet, since it needs the infusion point VDF. Once the infusion_iterations VDF has been released, it is added to `b1'` to form the finished block at `b1`.

Recall that in this example,

- sub-slot_iterations = 100M
- sp_interval_iterations is 1.5625M. Furthermore, let's say a farmer has a total of 1000 plots.

For each of the 64 signage points, as they are released to the network every 9.375 seconds, or every 1.5625M iterations, the farmer computes the plot filter and sees how many plots pass. For each passing plot, the farmer calculates required_iterations.

Let's say the farmer calculates required_iterations < 1.5625M once in the sub-slot. (We'll assume the exact required_iterations = 0.7828M in this instance.) Figure 5 shows this happening at the 20th signage point.

infusion_iterations is then computed as:

infusion_iterations = signage_point_iterations + (3 * sp_interval_iterations) + required_iterations

= (signage * point * sp * interval_iterations) + (3 * sp_interval_iterations) + required_iterations

= (20 * 1.5625M) + (3 * 1.5626M) + 0.7827M

= 36.7223M

After realizing they have won (at the 20th infusion point), the farmer fetches the whole proof of space, makes a beacon block (optionally including execution payload), and broadcasts this to the network. The block has until infusion_iterations (typically a few seconds) to reach timelords, who will infuse the block, creating the infusion point VDFs. With these VDFs, the block can be finished and added to the beacon chain by other beacon clients.

Rationale for choosing 64 signage points

The maximum distance between a signage point and the next infusion point is 4 signage points (see the `infusion_iterations` formula, above), or 37.5 seconds. This is the maximum amount of time for the replotting attack to be possible, but for it to be consistently applied, the minimum time of 28.125 seconds must be applied. Assuming a few extra seconds for network latency and other factors, the attack is now only possible if one can create a new plot in less than 25 seconds.

This begs the question: why not use even more signage points in the consensus? The simple answer is because as the signage points increase, it becomes more difficult for the timelords and nodes to keep up with the network. Sixty-four signage points per subslot was deemed enough to discourage the attack laid out above, while still allowing the timelords and nodes to perform as intended and to stay in sync.

Revision #3
Created 5 June 2023 14:41:34 by Admin
Updated 6 June 2023 06:56:04 by Admin