Toward a 12-second Block Time

Libreta:	Primera Libreta
Creado:	09/07/2018 21:22
URLOrigen:	https://blog.ethereum.org/2014/07/11/toward-a-12-second-block-time/

Toward a 12-second Block Time

Posted by Vitalik Buterin on July 11, 2014

One of the annoyances of the blockchain as a decentralized platform is the sheer length of delay before a transaction gets finalized. One confirmation in the Bitcoin network takes ten minutes on average, but in reality due to statistical effects when one sends a transaction one can only expect a confirmation within ten minutes 63.2% of the time; 36.8% of the time it will take longer than ten minutes, 13.5% of the time longer than twenty minutes and 0.25% of the time longer than an hour. Because of fine technical points involving Finney attacks and sub-50% double spends, for many use cases even one confirmation is not enough; gambling sites and exchanges often need to wait for three to six blocks to appear, often taking over an hour, before a deposit is confirmed. In the time before a transaction gets into a block, security is close to zero; although many miners refuse to forward along transactions that conflict with transactions that had already been sent earlier, there is no economic necessity for them to do so (in fact quite the contrary), and some don't, so reversing an unconfirmed transaction is possible with about a 10-20% success rate.

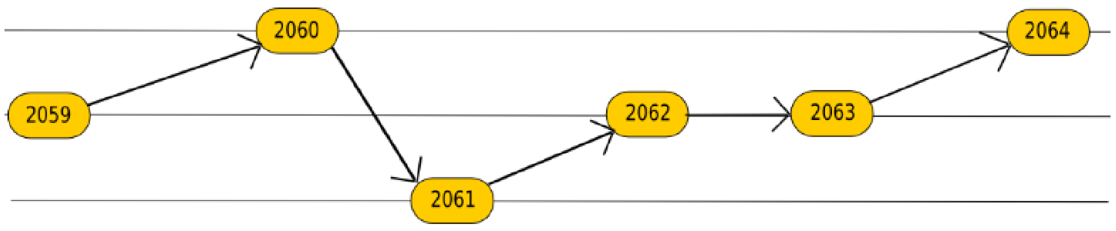
In many cases, this is fine; if you pay for a laptop online, and then manage to yank back the funds five minutes later, the merchant can simply cancel the shipping; online subscription services work the same way. However, in the context of some in-person purchases and digital goods purchases, it is highly inconvenient. In the case of Ethereum, the inconvenience is greater; we are trying to be not just a currency, but rather a generalized platform for decentralized applications, and especially in the context of non-financial apps people tend to expect a much more rapid response time. Thus, for our purposes, having a blockchain that is faster than 10 minutes is critical. However, the question is, how low can we go, and if we go too low does that destabilize anything?

Overview of Mining

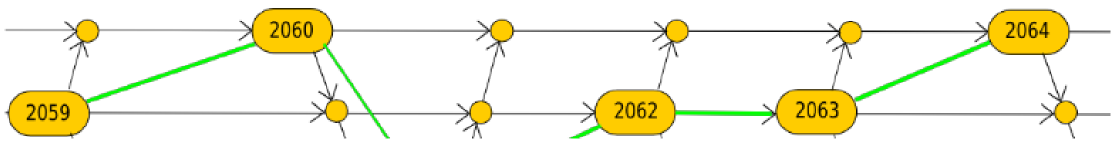
First off, let us have a quick overview of how mining works. The Bitcoin blockchain is a series of blocks, with each one pointing to (ie. containing the hash of) the previous. Each miner in the network attempts to produce blocks by first grabbing up the necessary data (previous block, transactions, time, etc), building up the block header, and then continually changing a value called the nonce until the nonce satisfies a function called a "proof of work condition" (or "mining algorithm"). This algorithm is random and usually fails; on average, in Bitcoin the network needs to collectively make about 10^{20} attempts before a valid block is found. Once some random miner finds a block that is valid (ie. it points to a valid previous block, its transactions and metadata are valid, and its nonce satisfies the PoW condition), then that block is broadcast to the network and the cycle begins again. As a reward, the miner of that block gets some quantity of coins (25 BTC in Bitcoin) as a reward.

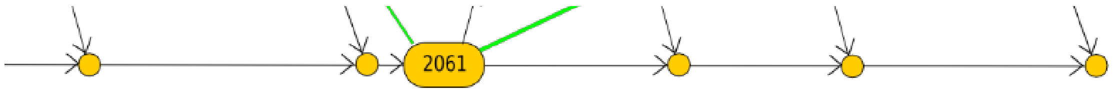
The "score" of a block is defined in a simplified model as the number of blocks in the chain going back from it all the way to the genesis (formally, it's the total mining difficulty, so if the difficulty of the proof of work condition increases blocks created under this new more stringent condition count for more). The block that has the highest score is taken to be "truth". A subtle, but important, point is that in this model the incentive for miners is always to mine on the block with the highest score, because the block with the highest score is what users ultimately care about, and there are never any factors that make a lower-score block better. If we fool around with the scoring model, then if we are not careful this might change; but more on this later.

We can model this kind of network thus:

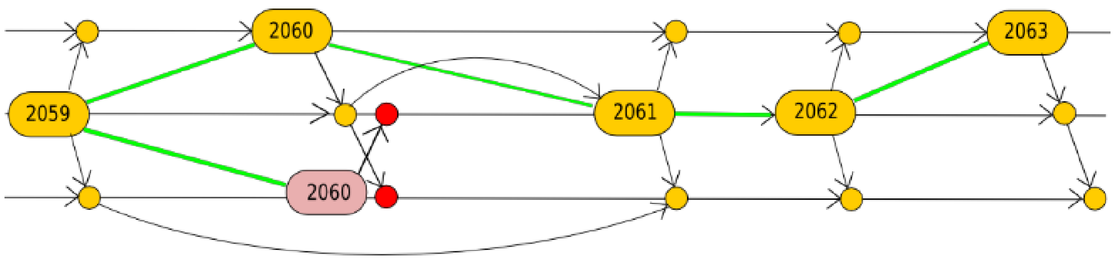


However, the problems arise when we take into account the fact that network propagation is not instant. According to a 2013 paper from Decker and Wattenhofer in Zurich, once a miner produces a block on average it takes 6.5 seconds for the block to reach 50% of nodes, 40 seconds for it to reach 95% of nodes and the mean delay is 12.6 seconds. Thus, a more accurate model might be:





This gives rise to the following problem: if, at time T = 500, miner M mines a block `B'` on top of `B` (where "on top of" is understood to mean "pointing to as the previous block in the chain"), then miner N might not hear about the block until time T = 510, so until T = 510 miner N will still be mining on B. If miner B finds a block in that interval, then the rest of the network will reject miner B's block because they already saw miner A's block which has an equal score:



Stales, Efficiency and Centralization

So what's wrong with this? Actually, two things. First, it weakens the absolute strength of the network against attacks. At a block time of 600 seconds, as in Bitcoin, this is not an issue; 12 seconds is a very small amount of time, and Decker and Wattenhofer estimate the total stale rate as being around 1.7%. Hence, an attacker does not actually need 50.001% of the network in order to launch a 51% attack; if the attacker is a single node, they would only need `0.983 / 1 + 0.983` = 49.5%. We can estimate this via a mathematical formula: if transit time is 12 seconds, then after a block is produced the network will be producing stales for 12 seconds before the block propagates, so we can assume an average of `12 / 600 = 0.02` stales per valid block or a stale rate of 1.97%. At 60 seconds per block, however, we get `12 / 60 = 0.2` stales per valid block or a stale rate of 16.67%. At 12 seconds per block, we get `12 / 12 = 1` stale per valid block, or a stale rate of 50%. Thus, we can see the network get substantially weaker against attacks.

However, there is also another negative consequence of stale rates. One of the more pressing issues in the mining ecosystem is the problem of mining centralization. Currently, most of the Bitcoin network is split up into a small number of "mining pools", centralized constructions where miners share resources in order to receive a more even reward, and the largest of these pools has for months been bouncing between 33% and 51% of network hashpower. In the future, even individual miners may prove threatening; right now 25% of all new bitcoin mining devices are coming out of a single factory in Shenzhen, and if the pessimistic version of my economic analysis proves correct that may eventually morph into 25% of all Bitcoin miners being in a single factory in Shenzhen.

So how do stale rates affect centralization? The answer is a clever one. Suppose that you have a network with 7000 pools with 0.01% hashpower, and one pool with 30% hashpower. 70% of the time, the last block is produced by one of these miners, and the network hears about it in 12 seconds, and things are somewhat inefficient but nevertheless fair. 30% of the time, however, it is the 30% hashpower mining pool that produced the last block; thus, it "hears" about the block instantly and has a 0% stale rate, whereas everyone else still has their full stale rate.

Because our model is still pretty simple, we can still do some math on an approximation in closed form. Assuming a 12 second transit time and a 60-second block time, we have a stale rate of

16.67% as described above. The 30% mining pool will have a 0% stale rate 30% of the time, so its efficiency multiplier will be `0.833 * 0.7 + 1 * 0.3 = 0.8831` , whereas everyone else will have an efficiency multiplier of 0.833; that's a 5.7% efficiency gain which is pretty economically significant especially for mining pools where the difference in fees is only a few percent either way. Thus, if we want a 60 second block time, we need a better strategy.

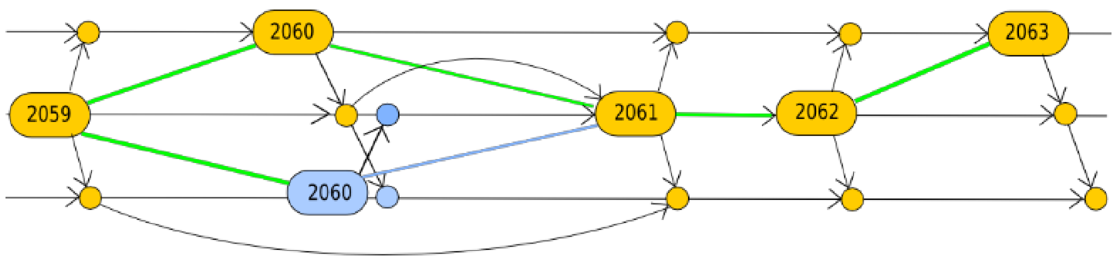
GHOST

The beginnings of a better approach come from a paper entitled "Fast Money Grows on Trees, not Chains", published by Aviv Zohar and Yonatan Sompolinsky in December 2013. The idea is that even though stale blocks are not currently counted as part of the total weight of the chain, they could be; hence they propose a blockchain scoring system which takes stale blocks into account even if they are not part of the main chain. As a result, even if the main chain is only 50% efficient or even 5% efficient, an attacker attempting to pull off a 51% attack would still need to overcome the weight of the entire network. This, theoretically, solves the efficiency issue all the way down to 1-second block times. However, there is a problem: the protocol, as described, only includes stales in the scoring of a blockchain; it does not assign the stales a block reward. Hence, it does nothing to solve the centralization problem; in fact, with a 1-second block time the most likely scenario involves the 30% mining pool simply producing *every* block. Of course, the 30% mining pool producing every block *on the main chain* is fine, but only if the blocks off chain are also fairly rewarded, so the 30% mining pool still collects not much more than 30% of the revenue. But for that rewarding stales will be required.

Now, we can't reward all stales always and forever; that would be a bookkeeping nightmare (the algorithm would need to check very diligently that a newly included uncle had never been included before, so we would need an "uncle tree" in each block alongside the transaction tree and state tree) and more importantly it would make double-spends cost-free. Thus, let us construct our first protocol, single-level GHOST, which does the minimal thing and takes uncles only up to one level (this is the algorithm used in Ethereum up to now):

1. Every block must point to a parent (ie. previous block), and can also include zero or more uncles. An "uncle" is defined as a block with a valid header (the block itself need not be valid, since we only care about its proof-of-work) which is the child of the parent of the parent of the block but not the parent (ie. the standard definition of "uncle" from genealogy that you learned at age 4).
2. A block on the main chain gets a reward of 1. When a block includes an uncle, the uncle gets a reward of 7/8 and the block including the uncle gets a reward of 1/16.
3. The score of a block is zero for the genesis block, otherwise the score of the parent plus the difficulty of the block multiplied by one plus the number of included uncles.

Thus, in the graphical blockchain example given above, we'll instead have something like this:



Here, the math gets more complex, so we'll make some intuitive arguments and then take the lazy approach and simulate the whole thing. The basic intuitive argument is this: in the basic mining protocol, for the reasons we described above, the stale rate is roughly `t/(T+t)` where `t` is the transit time and `T` is the block interval, because `t/T` of the time miners are mining on old data. With single-level GHOST, the failure condition changes from mining one stale to mining two stales in a row (since uncles can get included but relatives with a divergence of 2 or higher cannot), so the stale rate should be `(t/T)^2`, ie. about 2.7% instead of 16.7%. Now, let's use a Python script to test that theory:

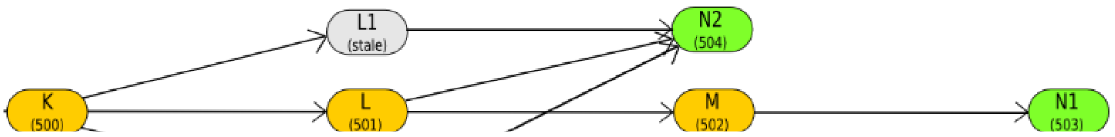
```
### PRINTING RESULTS ###
1 1.0
10 10.2268527074
25 25.3904084273
5 4.93500893242
15 14.5675475882

Total blocks produced:  16687
Total blocks in chain:  16350
Efficiency:  0.979804638341
Average uncles:  0.1584242596
Length of chain:  14114
Block time:  70.8516366728
```

The results can be parsed as follows. The top five numbers are a centralization indicator; here, we see that a miner with 25% hashpower gets 25.39x as much reward as a miner with 1% hashpower. The efficiency is 0.9798 meaning that 2.02% of all blocks are not included at all, and there are 0.158 uncles per block; hence, our intuitions about a ~16% stale rate without uncle inclusion and 2.7% with uncle inclusion are confirmed almost exactly. Note that the actual block time is 70.85s because even though there is a valid proof of work solution every 60s, 2% of them are lost and 14% of them make it into only the next block as an uncle, not into the main chain.

Now, there is a problem here. The original authors of the GHOST paper did not include uncle/stale rewards, and although I believe it is a good idea to deviate from their prescription for the reasons I described above, they did not do so for a reason: it makes the economic analysis more uncomfortable. Specifically, when only the main chain gets rewarded there is an unambiguous argument why it's always worth it to mine on the head and not some previous block, namely the fact that the only thing that conceivably differentiates any two blocks is their score and higher score is obviously better than lower score, but once uncle rewards are introduced there are other factors that make things somewhat tricky.

Specifically, suppose that the main chain has its last block `M` (score 502) with parent `L` (score 501) with parent `K` (score 500). Also suppose that `K` has two stale children, both of which were produced after `M` so there was no chance for them to be included in `M` as uncles. If you mine on `M`, you would produce a block with score `502 + 1 = 503` and reward 1, but if you mine on `L` you would be able to include `K`'s children and get a block with score `501 + 1 + 2 = 504` and reward `1 + 0.0625 * 2 = 1.125`.



Additionally, there is a selfish-mining-esque attack against single-level GHOST. The argument is as follows: if a mining pool with 25% hashpower were not to include any other blocks, then in the short term it would hurt itself because it would no longer receive the 1/16x nephew reward but it would hurt others more. Because in the long-term mining is a zero-sum game since the block time rebalances to keep issuance constant, this means that not including uncles might actually be a dominant strategy, so centralization concerns are not entirely gone (specifically, they still remain 30% of the time). Additionally, if we decide to crank up the speed further, say to a 12 second target block time, single-level is just not good enough. Here's a result with those statistics:

```
### PRINTING RESULTS ###
1 1.0
10 10.4567533177
15 16.3077390517
5 5.0859101624
25 29.6409432377

Total blocks produced:  83315
Total blocks in chain:  66866
Efficiency:  0.802568565084
Average uncles:  0.491246459555
Length of chain:  44839
Block time:  22.3020138719
```

18% centralization gain. Thus, we need a new strategy.

A New Strategy

The first idea I tried about one week ago was requiring every block to have five uncles; this would in a sense decentralize the production of each block further, ensuring that no miner had a clear advantage in making the next block. Since the math for that is pretty hopelessly intractable (well, if you try hard at it for months maybe you could come up with something involving nested Poisson processes and combinatorical generating functions, but I'd rather not), here's the sim script. Note that there are actually two ways you can do the algorithm: require the parent to be the lowest-hash child of the grandparent, or require the parent to be the highest-score child of the grandparent. The first way (to do this yourself, modify line 56 to `if newblock["id"] > self.blocks[self.head]["id"]:` , we get this:

```
### PRINTING RESULTS ###
1 1.0
10 9.59485744106
25 24.366668248
5 4.82484937616
15 14.0160823568

Total blocks produced:  8033
Total blocks in chain:  2312
Efficiency:  0.287812772314
Average uncles:  385.333333333
Length of chain:  6
Block time:  13333.3333333
```

Ooooops! Well, let's try the highest-score model:

```
### PRINTING RESULTS ###
1 1.0
10 9.76531271652
15 14.1038046954
5 5.00654546181
25 23.9234131003

Total blocks produced:  7989
Total blocks in chain:  6543
Efficiency:  0.819001126549
Average uncles:  9.06232686981
Length of chain:  722
Block time:  110.8033241
```

So here we have a very counterintuitive result: the 25% hashpower mining pool gets only 24x as much as a 1% hashpower pool. Economic sublinearity is a cryptoeconomic holy grail, but unfortunately it is also somewhat of a perpetual motion machine; unless you rely on some specific thing that people have a certain amount of (eg. home heating demand, unused CPU power), there is no way to get around the fact even if you come up with some clever sublinear concoction an entity with 25x as much power going in will at the very least be able to pretend to be 25 separate entities and thus claim a 1x reward. Thus, we have an unambiguous (okay, fine, 99 point something percent confidence) empirical proof that the 25x miners are acting suboptimally, meaning that the optimal strategy in this environment is not to always mine the block with the highest score.

The reasoning here is this: if you mine on a block that has the highest score, then there is some chance that someone else will discover a new uncle one level back, and then mine a block on top of that, creating a new block at the same level as your block but with a slightly higher score and leaving you in the dust. However, if you try to be one of those uncles, then the highest-score block at the next level will certainly want to include you, so you will get the uncle reward. The presence of one non-standard strategy strongly suggests the existence of other, and more exploitative, non-standard strategies, so we're not going this route. However, I chose to include it in the blog post to show an example of what the dangers are.

So what is the best way forward? As it turns out, it's pretty simple. Go back to single level GHOST, but allow uncles to come from up to 5 blocks back. Hence, the child of a parent of a parent (hereinafter, -2,+1-ancestor) is a valid uncle, a -3,+1-ancestor is a valid uncle, as is a -4,+1-ancestor and a -5,+1-ancestor, but a -6,+1-ancestor or a -4,+2-ancestor (ie. `c(c(P(P(P(P(head))))))` where no simplification is possible) is not. Additionally, we increase the uncle reward to 15/16, and cut the nephew reward to 1/32. First, let's make sure that it works under standard strategies. In the GHOST sim script, set `UNCLE_DEPTH` to 4, `POW_SOLUTION_TIME` to 12, `TRANSIT_TIME` to 12, `UNCLE_REWARD_COEFF` to 15/16 and `NEPHEW_REWARD_COEFF` to 1/32 and see what happens:

```
### PRINTING RESULTS ###
1 1.0
10 10.1329810896
25 25.6107014231
5 4.96386947539
15 15.0251826297

Total blocks produced:  83426
Total blocks produced:  83426
Total blocks in chain:  77306
Efficiency:  0.926641574569
Average uncles:  0.693116362601
Length of chain:  45659
Block time:  21.901487111
```

Completely reasonable all around, although note that the actual block time is 21s due to inefficiency and uncles rather than the 12s we targeted. Now, let's try a few more trials for enlightenment and fun:

- `UNCLE_REWARD_COEFF = 0.998`, `NEPHEW_REWARD_COEFF = 0.001` lead to the 25% mining pool getting a roughly 25.3x return, and setting `UNCLE_REWARD_COEFF = 7/8` and `NEPHEW_REWARD_COEFF = 1/16` leads to the 25% mining pool getting a 26.26% return. Obviously setting the `UNCLE_REWARD_COEFF` all the way to zero would negate the benefit completely, so it's good to have it be as close to one as possible, but if it's too close to one than there's no incentive to include uncles. `UNCLE_REWARD_COEFF = 15/16` seems to be a fair middle ground, giving the 25% miner a 2.5% centralization advantage
- Allowing uncles going back 50 blocks, surprisingly, has fairly little substantial efficiency gain. The reason is that the dominant weakness of -5,+1 GHOST is the +1, not the -5, ie. stale `c(c(P(P(..P(head)..))))` blocks are the problem. As far as centralization goes, with 0.998/0.001 rewards it knocks the 25% mining pool's reward down to essentially 25.0x. With 15/16 and 1/32 rewards there is no substantial gain over the -4,+1 approach.
- Allowing -4,+3 children increases efficiency to effectively 100%, and cuts centralization to near-zero assuming 0.998/0.001 rewards and has negligible benefit assuming 15/16 and 1/32 rewards.

15/16 and 1/32 rewards.

- If we reduce the target block time to 3 seconds, efficiency goes down to 66% and the 25% miner gets a 31.5x return (ie. 26% centralization gain). If we couple this with a -50,+1 rule, the effect is negligible (25% -> 31.3x), but if we use a -4,+3 rule efficiency goes up to 83% and the 25% miner only gets a 27.5x return (the way to add this to the sim script is to add after line 65 `for c2 in self.children.get(c, {}): u[c2] = True` for a -n,+2 rule and then similarly nest down one level further for -n,+3). Additionally, the actual block time in all three of these scenarios is around 10 seconds.
- If we reduce the target block time to 6 seconds, then we get an actual block time of 15 seconds and the efficiency is 82% and the 25% miner gets 26.8x even without improvements.

Now, let's look at the other two risks of limited GHOST that we discussed above: the non-head dominant strategy and the selfish-mining attack. Note that there are actually two non-head strategies: try to take more uncles, and try to be an uncle. Trying to take more uncles was useful in the -2,+1 case, and trying to be an uncle was useful in the cas of my abortive mandatory-5-uncles idea. Trying to be an uncle is not really useful when multiple uncles are not required, since the reason why that alternative strategy worked in the mandatory-5-uncle case is that a new block is useless for further mining without siblings. Thus, the only potentially problematic strategy is trying to include uncles. In the one-block case, it was a problem, but here is it not because most uncles that can be included after `n` blocks can also be included after `n+1` blocks, so the practical extent to which it will matter is limited.

The selfish-mining attack also no longer works for a similar

The selfish mining attack also no longer works for a similar reason. If you fail to include uncles, then the guy after you will. There are four chances for an uncle to get in, so not including uncles is a 4-party prisoner's dilemma between anonymous players - a game that is doomed to end badly for everyone involved (except of course the uncles themselves). There is also one last concern with this strategy: we saw that rewarding all uncles makes 51% attacks cost-free, so are they cost-free here? Beyond one block, the answer is no; although the first block of an attempted fork will get in as an uncle and receive its 15/16x reward, the second and third and all subsequent ones will not, so starting from two confirmations attacks still cost miners almost as much as they did before.

Twelve seconds, really?

The most surprising finding about Decker and Wattenhofer's finding is the sheer length of time that blocks take to propagate - an amazingly slow 12 seconds. In Decker and Wattenhofer's analysis, the 12 second delay is actually mostly because of the need to download and verify the blocks themselves; ie. the algorithm that Bitcoin clients follow is:

```
def on_receive_block(b):
    if not verify_pow_and_header(b):
        return
    if not verify_transactions(b):
        return
    accept(b)
    start_broadcasting(b)
```

However, Decker and Wattenhofer did propose a superior strategy which looks something like this:

strategy which looks something like this:

```
def on_receive_header(h):
    if not verify_pow_and_header(h):
        return
    ask_for_full_block(h, callback)

    start_broadcasting(h)
    def callback(b):
        start_broadcasting(b)
        if not verify_transactions(b):
            stop_broadcasting(b)
            return
        accept(b)
```

This allows all of the steps to happen in parallel; headers can get broadcasted first, then blocks, and the verifications do not need to all be done in series. Although Decker and Wattenhofer do not provide their own estimate, intuitively this seems like it may speed up propagation by 25-50%. The algorithm is still non-exploitable because in order to produce an invalid block that passes the first check a miner would still need to produce a valid proof of work, so there is nothing that the miner could gain. Another point that the paper makes is that the transit time is, beyond a certain point, proportional to block size; hence, cutting block size by 50% will also cut transit time to something like 25-40%; the nonscaling portion of the transit time is something like 2s. Hence, a 3-second target block time (and 5s actual block time) may be quite viable. As usual, we'll be more conservative at first and not take things that far, but a block time of 12s does nevertheless seem to be very much achievable.

← PREVIOUS POST

NEXT POST →

[illegible]

E [illegible]