Analyzing the ECO public chain performance and its special characteristics

Akis Chalkidis akis@ecoc.io

October 2018

Contents

1 Preface

In this paper we are going to analyze the characteristics of ECO chain. More specificaly, we are going to speak about transaction speed, the scalability problem, mean and variance of block creation time , possible ways to attack the chain, orphaned blocks and network efficiency , formal verification and many more.

2 Transaction speed

Of particular interest for a chain which plans to host many Dapps is the capasity of the chain; that is, how many transaction per second (TpS) it can receive. Here we must separate two variables: the maximum TpS and the sustainable TpS, that is, the average TpS in the long run which can keep the chain usable.

Let's analyse the maximum TpS first. We are going to show on which parameteres it depends and what restrictions exist in real world and put a limit to the transaction speed.

2.1 Maximum TpS

Now, let:
h be the number of hops
b be the bandwidth in bits per second
s be the maximum block size in bytes
n be the number of nodes
c be the number of outbound connections for each node
ts be the desired transaction speed (TpS)
l be the average transaction size in bytes

First, we are going to compute the number of hops required until all the nodes get the data. Because the number of connections is fixed, c , and because each node that has the data (block) broadcasts to the other nodes that it is connected, the flow is following a "snowball" effect. This, in mathematics is a geometric progression:

$$a_h = a_0 c^{h-1}$$

where a_h is the number of nodes that are informed after $h-1$ hopes. Here we assume that $a_0 = 1$, because a_0 is the node who forms(wins) the block and is ready to start broadcasting. The total informed nodes n after the h hops will be

$$\sum a_h = \frac{a_0(1-c^h)}{1-c} \Rightarrow \sum a_h = \frac{1-c^h}{1-c} \quad (1)$$

and for all nodes to be informed the hopes h can be computed from the inequality $\sum a_h \geq n$

$$\sum a_h \geq n \Rightarrow \frac{1-c^h}{1-c} \geq n \Leftrightarrow \frac{c^h-1}{c-1} \geq n$$

and because $c > 1 \Leftrightarrow c - 1 > 0$ we have:

$$\frac{c^h - 1}{c - 1} \geq n \quad \wedge \quad c > 1 \Rightarrow c^h \geq n(c-1) + 1 \Leftrightarrow \log_c c^h \geq \log_c (nc - n + 1) \Leftrightarrow$$

$$\Leftrightarrow h \geq \log_c (nc - n + 1)$$

We must keep in mind that the number of hops h is an integer and also the first hop is actualy happening for the term $a_1 \rightarrow a_2$ of the geometric progression. In short, $h = \Big\lfloor h \Big\rfloor + 1 - 1 \Leftrightarrow h = \Big\lfloor h \Big\rfloor \Rightarrow h = \Big\lfloor \log_c (nc - n + 1) \Big\rfloor$. So we finally have

$$h = \Big\lfloor \log_c (nc - n + 1) \Big\rfloor \quad (2)$$

We know h now , so we can continue to find what is the lowest bandwidth requirment b to achieve the desired transaction speed ts, where ts is the number of maximum transactions per second. Here we asume that the maximum blocksize s is very large compared to block header, that is , $\frac{bl}{s} \approx 0$, where bl is the length of the block header. Obviously, the maximum transactions per block are $tr \approx \frac{s}{l}$. Let's define bt as the target(average) block creation time in seconds. We have:

$$ts \geq \frac{tr}{bt}$$

The propogation time required for one "push", that is, one hop is :

$$T_i = T_l + T_p$$

where T_i is the total time in seconds required for each hop to complete , T_l is the latency and T_p the time needed to broadcast the blockdata. This time depends on the upload bandwidth of broadcasters and the download bandwidth of the recievers. Latency mainly depends on the location of the nodes (physical topology of the network). Let's asume for now that for large block size data, T_p is significantly larger than T_l, in other words assume that $T_i \approx T_p$. This is not very accurate but it will help the simplification of analysis.

For a hop h_i the maximum time $T_i \approx T_p = 8\frac{s}{b}$ because a byte equals 8 bits and s is measured in bytes. The total propogation time , let's say t_T is

$$t_T = \sum T_i \approx \sum T_p = \sum 8\frac{s}{b} = 8\sum \frac{s}{b}$$

We can make one more assumption here, that each hop h_i needs the same propogation time, because all the conditions are the same (average bandwidth, block size to be propogated). Consequently

$$t_T \approx 8\sum_{i=1}^{h} \frac{s}{b} = 8h\frac{s}{b}$$

We have already proved that $h = \Big\lfloor \log_c (nc - n + 1) \Big\rfloor$, so

$$t_T \approx 8h\frac{s}{b} \Rightarrow t_T \approx \frac{8s\Big\lfloor \log_c (nc - n + 1) \Big\rfloor}{b} \quad (3)$$

Computing the above approximation was not hard at all, but is beneficial: It clearly shows that the total time t_T depends on the maximum block size s , the number of connections c , the number of nodes n and the average network speed (bandwidth) b. So it is NOT depending on the block creation time bt. But bt sets an obvious limitation:

$$t_T < bt \quad (4)$$

And is very easy to see why. The nodes should have enough time to "get" (download) the block data. So , while it is tolerable for some nodes to get the data from previous blocks while new ones are created, this is not safe to carry on for long; total propogation time should be lower than the average creation time of a block.

Combining (3) and (4) we get

$$t_T \approx \frac{8s\left\lfloor \log_c(nc-n+1)\right\rfloor}{b} \Rightarrow bt > \frac{8s\left\lfloor \log_c(nc-n+1)\right\rfloor}{b} \Leftrightarrow$$

$$\Leftrightarrow b > \frac{8s\left\lfloor \log_c(nc-n+1)\right\rfloor}{bt} \Leftrightarrow$$

$$\Leftrightarrow b_{min} = \frac{8s\left\lfloor \log_c(nc-n+1)\right\rfloor}{bt} \quad (5)$$

Equality (5) shows clearly the lowest bandwidth b_{min} that the network must have to sustain a blocksize of size s in bytes. It is also worth noting that the propogation of data is easily scalable because there is a logarithmic relation between the number of nodes of the network n and the number of connections c. In other words, the number of hops $h(n)$ is $\mathcal{O}(\log n)$. The reason of this efficiency is that the propogation is based on the *gossip protocol* [1]. Bitcoin and Ecochain follow the gossip protocol with a (default) number of connections $c = 8$. That can be easily seen in the code in the file **src/net.h**:

```
static const int MAX_OUTBOUND_CONNECTIONS = 8;
```

From the above we conclude that for Ecochain $c = 8$. As a sidenote, we must stress the point that any node is free to change the outbound number of connections; that is, c is not a parameter of the consensus protocol. Just changing the above line of code from 8 to any number is acceptable by the network, as it is not really detectable. For example, a node may have a high upload bandwidth and also may want to help the network, so chooses to set c to 100. Or the node is selfish or malicious and sets c to zero, not broadcoasting anything. In short, changing c is an easy soft fork. We can safely assume here that the vast majority of nodes are not going to change c as they don't have an incentive to do so. If the default value of c is set to a value greater than 8 that does not necesserally brings faster propogation time; there are restrictions of disk writting time when the database is commited and of CPU delay because

of the needed validation time when the number of transactions is large. So Ecochain keeps $c = 8$, which as we already mentioned is not restricting for the nodes; each node can change it very easily.

Let's see an example in numbers: for a maximum block size $s = 4Mbytes$, block time $bt = 32sec$, number of nodes $n = 4,000$ and number of connections $c = 8$ the minimum network speed (bandwidth) from equeation (5) should be

$$b_{min} = \frac{8s\left\lfloor \log_c (nc - n + 1) \right\rfloor}{bt} = \frac{8 * 4000000 * \left\lfloor \log_8 (4000 * 8 - 4000 + 1) \right\rfloor}{32} \frac{bits}{sec} =$$

$$= 1000000 * \left\lfloor \log_8 28001 \right\rfloor \frac{bits}{sec} = 1000000 * \left\lfloor 4.92439691020751 \right\rfloor \frac{bits}{sec} =$$

$$= 1000000 * 4 \frac{bits}{sec} = 4000000 = 4Mbps$$

As we have already seen the function $b_{min}(n)$ is logarithmic. So with the same parametres but with a much larger number of nodes $n = 30,000$ for example we have

$$b_{min} = \frac{8s\left\lfloor \log_c (nc - n + 1) \right\rfloor}{bt} = \frac{8 * 4000000 * \left\lfloor \log_8 (30000 * 8 - 30000 + 1) \right\rfloor}{32} \frac{bits}{sec} =$$

$$= 1000000 * \left\lfloor \log_8 210001 \right\rfloor \frac{bits}{sec} = 1000000 * \left\lfloor 5.8933455574294 \right\rfloor \frac{bits}{sec} =$$

$$= 1000000 * 5 \frac{bits}{sec} = 5000000 = 5Mbps$$

Today's (Q4 2018) average global bandwidth is around 10Mbps. Also, we must keep in mind that as the telecommunication's technology advances the bandwidth will get higher. There is a belief that the increase in bandwidth follows *Nielsen's Law* [2], which states that global bandwidth increases around 50% every year. This prediction is somewhat optimistic, as historical data show a slower annual rate increase. But the truth is that the increase rate is significant. Consequently, even if some people think that 4Mbps or 5Mbps is marginally achieveable today it must be clear that it could be easily achieveable in the near future.

Let us finally compute the TpS for the above parameteres. For blocksize $s = 4MB$, blocktime $bt = 32seconds$ and asumming that the average length of a transaction in bytes is $l \approx 200$ we get a maximum transaction speed of

$$ts \geq \frac{tr}{bt} \approx \frac{\frac{s}{l}}{bt} = \frac{s}{l * bt} \approx \frac{4 * 10^6}{200 * 32} = \frac{10^6}{200 * 8} = \frac{10^6}{1600} \approx 625$$

So maximum TpS $ts \approx 625$ transactions/second. This is the speed in theory. In practice, we measured around 560 transactions per second. The tests were run on a low number of nodes ($n = 100$) on AWS cloud. A real network is far less ideal as there may be greater deviation of bandwidth, more often disconnects

etc. For this reason the network protocol has build-in capability to avoid long timeouts. If a node doesn't get a response or recieves very slow connection for longer than 2 seconds it disconnects and connects to another node; that way it avoids slow propagation time.

The reader must keep in mind that all the above is an approximation; we conclude these results under certain assumptions to simplify the analysis. For example, we assumed that the block header size is negligible compared to its body size (which is true). We also assumed that the latency time T_l is very short compared to the time to transmit the data T_p. This deserves more attention. The protocol in use is the TCP, which demands the well known "three-way handshake" (SYN,SYN-ACK,ACK). We assume here that the node connections are trans-continental, so the latency is usually between $80-120ms$ (miliseconds). Some connections would be intercontinental. For these few transactions the latency would be around $200ms$. If the network has technical problems (very often connections and disconnections of nodes) the latency will be greater, which may play a minor role in the final TpS result. For a network with large number of nodes a large scale latency (high latency infecting many nodes) is unprobabe; in practise it is not possible for most nodes to have connection problems. The third assumption we made is the average transaction size. The most common transaction has a vin and two vouts (one reciever and the "change" that return to the owner). This is usually 191 bytes long. But there will surely be transactions with more vouts. There is also the case of smart contracts(for example, when a long bytecode smart contract is deployed). It is really difficult to predict the average transaction size. What is measurable is the throughput and not the transactions. If the estimation of the average transaction size of $200bytes$ is optimistic then TpS may be somewhat lower.

A final note why we choose a high block size limit. The maximum size of the block can be set lower in the peer's code without any problem, in fact other peers have no way to know the size limit that has been set internally in each node. In sort, a soft fork can lower the blocksize just changing a line inside the code of **src/ecoc/ecoc.h** file:

```
const int blockSizeLimit = 4000000 ; // blocksize limit
```

The opposite is NOT true. If the node wants to increase the block size limit it can really not. Just increasing the size, mining and forming a block beyond the common accepted limit will lead to rejection from the other nodes, as it violates the consesus protocol rules. In other words, increasing the blocksize limit demands a hardfork; that is, everyone must update their client version so the change can take effect. This is an additional reason that we initially preffer a high block size limit.

2.2 Sustainable TpS

It is time to talk about the scalability issue in the economic sense. In the previous section we analysed the technical restrictions for blockchain's capacity. In this section we are going to analyse the restrictions that the real economy sets. There are real world costs to run a full node, which grows linearly following the total size of the blockchain. The data of the public chain grow with a pace. This pace depends on average transactions per second and is measured in bytes per second. If the size's growth per unit of time is high then three problems arise for the preservers of the chain(nodes):

(a) the initial download time may take too long

(b) much RAM is needed

(c) much storage (disk space) is needed

Let's see each of these problems closely. For **(a)**, the problem is that when a node wishes to join the public chain for the first time he must download all the data (history). If the size is very large and the node has low download speed then it may take long time (several days) until the initial download completes. This may discourage new nodes to join the public chain.

For **(b)**, because UTXO model needs RAM linear to the transactions number, a large chain demands from the nodes a capability of high RAM (memory) usage. RAM is expensive, much more expensive than disk storage. So, considering the memory swap option, it looks natural for a node to use virtual memory (disk space as RAM). All operating systems for desktops today (Windows, Linux, Mac and all *nix systems) offer this capability. But there is a pitfall: disk is much slower than memory. This fact brings the following problem: the staker needs more time to validate a block, so he must delay some seconds before he starts minting(staking). While this is a problem in PoW protocol it does not pose a real problem in PoS if the granularity is longer than the validation time; that is, it does not put the staker in a disadvantage. We can assume that this problem , (b), given that virtual memory will be used, is equivalent with (c), which we are going to present immediately.

(c) The growth rate of chain's size depends on the actual (average) transaction speed. For example , if the average transaction speed is 50 transactions per second, and asuming an average length of 200 bytes per transaction , then in one year the size data will by larger by (we do not count block header in the equation as it is very small compared to the body)

$$\Delta S = \overline{tps} * \bar{l} * t = 50 * 200 * 60 * 60 * 24 * 365 = 315360000000 bytes \approx 315 Gb$$

which is a considerable amount of data. So we must find the sustainable TpS that can keep the nodes in the network, preserving decentralization. As a side node here, because it was never mentioned before, we must stress the point that for the staker to get the opportunity to mint and gain some profit he must run a full node. So a staker has an incentive to join the network running a node

as far as his expected gains from minting are greater than his costs(rational economic behavior). The costs are the hardware (basically the disk space) and the telecommunication costs. The decisive factor here is the cost of disk space. As we have already explained the RAM restriction resources can be converted to disk space resources. While we can not predict exactly the future costs or the staker's profits - which depend on coin's price - we can try to find the connection between the chain's size growth rate with relevance to the declining costs of disk storage as technology advances. That is , we can , under some assumptions, find a cost function $c(c_0, y)$, the value of which must be lower than the maximum acceptable cost for the staker, lets say c_{max}. Here c_0 is the cost at the time the staker joins the network and y the years after the time that the cost is c_0. So our first restriction is:

$$c_{max} > c(c_0, y) \quad (6)$$

Now, let:
$c_{max,i}$ be the maximum acceptable cost for the staker i(let's say in USD)
c_0 be the cost to run a node for at first join time (USD)
s_0 the minimum disk size for the node to run at starting year in GBytes
dc the cost of disk storage (in USD/Gbyte)
s_y the minimum disk size for the node to run at year y in GBytes (y years after joining)
gr the growth rate of the chain (measured in GBytes/year)

Obviously $gr = \overline{tps} * \bar{l} * (60 * 60 * 24 * 365)$, where $\overline{tps}$ stands for average transaction speed and $\bar{l}$ for average transaction length. We also consider that $\frac{b_h}{b_b} \approx 0$, where b_h is the blockheader size and b_b the body size. Obviously, $c(c_0, 0) = c_0$ by definition

The new cost after a year for the node will be

$$c_1 = c_0 - \delta c_{y_0->y_1} + \delta s_{y_0->y_1} * dc_1$$

c_1 is the cost to run a node after on year of first join (in USD)
$\delta c_{y_0->y_1}$ is the declined cost becase of technologic advance - that is, cost storage lowers over time.
$\delta s_{y_0->y_1}$ is the extra chain data size after a year
dc_1 the cost of disk storage (in USD/Gbyte) at y_1
The new maximum accepted cost is $c_{max,1}$ which can be higher or lower than $c_{max,0}$ as the price of the coin can be lower or the staker's balance may be lower giving him lower expected staking rewards.We are going to symbolize the annyally cost differnce and maximum accepted cost difference at year y as $\delta c_{y,y-1}$ and $\delta c_{max_{y,y-1}}$ perspectively.

The first think to notice is that stakers who have a very low balance (close to zero) do not have an economic motivation to run a full node. That is, at any time y, for a staker with a total current balance b_i, staking reward r per year and an exchange rate e between the coin and USD , to run a node the following inequality appears:

$$\delta c_{y,y-1} < \delta pr_{y,y-1} \quad (7)$$

where $pr = b_i * r * e$ is the staker's profit from minting. In plain english , the annual profit must cover the extra cost added per year. While it is clear that the profit is depending on the exchange rate e we must not conclude that is depending on the staking reward r because e is not independent from r; $e(r)$ is a monotonic declining function because r creates inflation driving the exchange rate down. In other words a higher value of r does not guarentee higher profits for the staker. As a side note, $\delta c_{y,y-1}$ may be negative. This may seem strange at first glance; how the cost for a node can decline when the chain data increase? But if a technology breakthrough take place, the storage cost dc_y will fall dramatically. In this case, if

$$\delta c_{y-1\to y} > \delta s_{y-1\to y} * dc_y$$

then

$$c_y = c_{y-1} - \delta c_{y-1\to y1} + \delta s_{y-1\to y} * dc_y \wedge \delta c_{y-1\to y} > \delta s_{y-1\to y_y} * dc_y \Rightarrow$$

$$\Rightarrow c_y < c_{y-1} \Leftrightarrow c_y - c_{y-1} < 0 \Leftrightarrow \delta c_{y-1\to y} < 0$$

So it is clear that the total cost may even decrease with the pass off time while chain size increases. In this case, even if the yearly profit decreases, but less than the cost annual reduction, it is still profitable for the staker to stay and run a full node. In short, the important factor is the annual **marginal** cost and the annual **marginal** profit from minting. By marginal here we mean the annual difference in values.

Let us be more precise. The staker does not know his future reward nor the future cost. So we are talking about the **expected** annual profit and **expected** annual cost. From the economic theory we know that for a rational economic behavior the marginal cost should be less or at least equal to the marginal profit (after the first initial investment in equipment). So the follow inequality holds for a stakholder:

$$M\overline{c_y} < M\overline{pr_y} \quad (8)$$

The marginal annual profit for a staker i $M\overline{pr_{y,i}}$ depends, as we have already seen, on his balance b_{y_i}, the current exchang rate e_y and the annual staking reward r. On the other hand the annual marginal cost depends on the growth rate of the public chain gr and the new cost per unit of disc storage dc_y. So the above inequality tranforms to

$$M\overline{c(gr, dc_y)} < M\overline{pr(b_{y_i}, r, e_i)} \quad (9)$$

Being more analytic for $M\overline{c(gr, dc_y)}$ we have:

$$M\overline{c(gr, dc_y)} = c_y - c_{y-1} \wedge c_y = c_{y-1} - \delta dc * s_{y-1} + \delta s_y * dc_y \Rightarrow$$

$$\Rightarrow M\overline{c(gr, dc_y)} = \delta s_y * dc_y - \delta dc * s_{y-1} \Leftrightarrow$$

$$M\overline{c(gr, dc_y)} = gr * dc_y - \delta dc * s_{y-1} \Leftrightarrow$$

gr is the growth rate of the chain, dc_y the storage cost at year y, δdc is the technological cost decrease per unit storage and s_{y-1} is the storage size of teh previous year. So finally we have:

$$M\overline{pr(b_{y_i}, r, e_i)} > gr * dc_y - \delta dc * s_{y-1} \quad (10)$$

For every rational staker to keep staking, that is, continue running a full node, the above inequality must hold. So his decision is based on his stake (his account balance), the exchange rate and annualy reward on staking, the growth rate of the chain and the decrease cost factor of disk storage (which depends on the technology evolving). Unfortunately , we can not make more simplifications as this may lead us to unaccurate results.

Let's see an example. Suppose that the growth of the chain was for the last year $gr = 315Gb$ (like our previous example), the cost of 1Gb on the current moment (year) is $dc * 1GB = \$0.04$, the cost of the previous year was $dc * 1GB = \$0.05$ and the total size of the chain the previous year was, let's say, $s_{y-1} = 1.5Tb$. So his marginal cost for the last year is

$$M\overline{c(gr, dc_y)} = gr * dc_y - \delta dc * s_{y-1} \Rightarrow$$

$$\Rightarrow M\overline{c(gr, dc_y)} = 315Gb * 0.04\$/Gb - (0.05\$ - 0.04\$)\$/Gb * 1.5 * 1000 \Leftrightarrow$$

$$\Leftrightarrow M\overline{c(gr, dc_y)} = -2.4\$USD$$

In the present example we see that the extra annual cost is negative, so it is lower from the last year's cost by 2.4$ So extra profit for the last year can be lower but no more than 2.4$:

$$M\overline{pr(b_{y_i}, r, e_i)} > -2.4\$ \Rightarrow pr_y - pr_{y-1} > -2.4\$ \Leftrightarrow pr_y + 2.4\$ > pr_{y-1}$$

Usualy the marginal cost will be positive, except in the case of a great unexpected tech innovation in storage or in the case that the growth rate of the data of the chain will be too low. In any case , the extra profit should cover the extra cost.

Let as check, for the above example, the maximum cost for a staker to join the network for first time. We have $c_{max,0} = s_0 * dc_0 \Rightarrow c_{max,0} = 1.5Tb * 0.05USD/Gb \Leftrightarrow c_{max,0} = 1.5*1000*0.05USD \Leftrightarrow c_{max,0} = 75USD \Rightarrow pr_{min,0} = 75USD$. So his expected profit should be more than $75USD$ to decide running a node.

In our analysis so far we have excluded the networking costs. This simplification makes our job simpler but more innaccurate. The truth is that for many machines the user pays for network traffic either way. In this case his network cost is zero. A typical example is a desktop user who already uses internet for his personal use or a server who is running doing various other things and has much bandwidth unused. There is also an "irrational behavior" in real world; a staker may have enough balance to his account but decide not to run a node or the opposite, to decide running a node even if he sufers a minor loss. Also, our

equations and inequalities depend on future exchange rates which are impossible to predict. In this section we have just shown mathmatically the relation between the factors who play a major role to a rational decisioner if and when he will join the chain or when he will have incentive to stop running a node.

What we must keep in mind is that decentralization depends on the number of nodes, which in turn depend on the value of the coin (the higher, the more nodes join) and the storage cost decreasing rate (again,the higher the decreasing cost rate, the more nodes join). There is *"Kryder's Law"* [3], which is not actually a law but a prediction of the disk storage costs for the year 2020. The prediction is somewhat about 40% decreasing rate per year. Until now his prediction has been seen as overoptimistic, but the fact is that the storage cost is really decreasing, although with an unpredictable pace. It is not unlike that somewhere in the future a sharp decline of cost may arise, for example a great innovation in the field of quantum storage or DNA use.

The takeover is that growth rate of the chain is allowed to increase the cost for the staker no more than his marginal (extra) profit. In this section we proved that this greatly depends on the declining storage costs.

3 Block creation time

Let's describe in short how PoS architecture reaches consesus. It is an imitation of the PoW proccess. The diffeence is that in PoS the "miner" can not pass any value(argument) to the hash function, as it is the case in PoW. He can only pass time (timestamp) and his public address (which must have a positive account balance). Off course, until the formation of the next block transcations can not take place, so public address is also fixed. His only option is to change the timestamp. When the timestamp changes, a hash is produced. The neccessary condition to win the block is :

$$h(c1, c2, ..., cn, ts, pa) < t * b$$

where :
h is result of the hash function
c1,c2,...,cn are arguments that cant be changed
ts is the timestamp (the only thing that staker can change)
pa is his public address
t is a value (target) that is set by the difficulty (like PoW) and got reset every fixed number of blocks
b is the balance in his public adress.

When the above inequality is true the stakeholder can claim and form the next block. The probability for each individual stakeholder to win the block at each different values of timestamp is $p_i = p * b_i$ where p_i is the probability of the individual to win the block proportionally to the balance b_i he has in his account. The probability at each timestamp (second), let's say each "tick", for

the total network to form a block is of course

$$P = p_1 + p_2 + p_3 + ... + p_s = \sum_1^s p_i = \sum_1^s p * b_i = p \sum_1^s b_i = p * B$$

where s subscript is the individual stakeholder (identified by his public address), B is the total balance of all stakeholders that participate in the consensus, that is, try to form ("win") the next block.

Until now, we didn't discuss how much is the probabilty p, so to be able to calculate the final probability P which the chain has to form a block at next "tick". Additionally, we are talking about the next tick and not the next second. True, we can force each stakeholder to try not every second but every $g+1$ seconds; we are going to call g as granularity from now on. $g+1$ is integer of course, and not only that but it has the form of $g + 1 = 2^k \Leftrightarrow g = 2^k - 1$, where k is also integer. For $k = 0$ we have the maximum granularity, that is, $g = 0$, which means that the staker can submit a hash each second.

First, we are going to give more details of how granularity works. And after that, we are going to explain why granularity adds security for the chain and what is the trade-off. To force the stakers to compute a hash every $g+1$ seconds and not every second we must mask the timestamp. This can be easily done if a NAND logic operation is used with timestamp. For example, if we want 4 sequental timestamps to give the same hash (rendering the 3 timestamp useless, as it they will give the same hash as the imput willl be the same) we can just do an AND operation of the last two bits of the timstamp with 0 (that is, binary xb00). So , doing a NAND with binary 11 (that is, 3) we mask the timestamps (setting the last two bits to zero) , transforming all 4 timestamps to the same value. Below we can see how this is implemented in the code(C++):

```
nTimeBlock &= ~STAKE_TIMESTAMP_MASK;
```

It is clear why granularity has the form $2^n - 1$. It is to mask the last n bits of the timestamp.

Low granularity (high g values) helps to prevent stake-grinding attacks. It is much harder for the attacker to perform a succesful attack. The implications of low granularity is higher variation for the block creation time. We are going to examine the mean creation time and variation of block time.

It is easy to see which probability distribution our model fits. Stakers can try to form a block every $g+1$ seconds , where $g \in \{0, 1, 3, 7, 15, 31, \ldots, 2^n-1\}$. $g+1$ is a power of 2. So every $g+1$ seconds they try to form a block. We are intresting for the mean time and cumulative distribution (CDF) of stakers to form a block, so we must calculate the p probabilty for a block to be formed. The tries make our distribution model *discrete* and not continous (as it is in bitcoin or any other PoW consensus). The correct probability distribution to pick is the **geometric distribution** [5] or **GD** [5] for short. Under the assumption that for each try the total stacking balance is about the same and the difficulty changes to

readjust the mean time to be fixed (target time), we asume that we already know the mean value ($mean = target\ time$). The geometric distribution is a discrete probability distrubution which computes the distrubution of X *Bernoulli trials* needed to get the first success after i number of tries. Keeping in mind that we have g and $target$ already known, and claryfing which distribution model describes exactly our model we can calculate the probability p, the cumulative probability to form a block after i tries and more.

We know from mathematichs that for **GD** the mean is:

$$E[p] = \frac{1}{p}$$

So we have: $E[p] = \frac{1}{p} \Rightarrow \frac{target}{g+1} = \frac{1}{p} \Leftrightarrow p = \frac{g+1}{target}$

The first occurence of a success attempt, that is, the creation of block after i tries is

$$P(X = i) = (1-p)^{i-1}p$$

For example, the probability for $g = 7$, $target = 128$ sec and $i = 1$ (first try) is:

$$P(X = i) = (1-p)^{i}p \Rightarrow P(X = 1) = (1 - 0.0625)^{1-1} * 0.0625 \Leftrightarrow$$

$$\Leftrightarrow P(X = 1) = 1 * 0.0625 \Leftrightarrow P(X = 1) = 6.25\%$$

Now,let $k = \frac{target}{g+1}$ We are intresting in the case where $i \leq k$, because this is where the average block time stands, or where the block must be created most of the time. Much longer than that is, obviously, less desirable. The probability that the block will be formed until the target time can be easily computed from the cumulative distribution function (CDF) for **G**D. We have

$$F_X(x) = P(X \leq x) \Rightarrow F_X(k) = 1-(1-p)^k \Rightarrow F_X(k) = 1-(1-\frac{g+1}{target})^{\frac{target}{g+1}} \Leftrightarrow$$

$$\Leftrightarrow F_X(k) = 1 - (\frac{target - g - 1}{target})^{\frac{target}{g+1}}$$

For the above example ($g = 7$, $target = 128$) $k = \frac{target}{g+1} = \frac{128}{7+1} = 16$ and

$$F_X(16) = 1 - (\frac{128 - 7 - 1}{128})^{\frac{128}{7+1}} = 1 - (\frac{120}{128})^{16} \approx 64.39\%$$

So we expect the block to be formed at the average time or before with a probability of 65.6%

Before we move to the next section to study the effectiveness and security we can make some observations from the table below, which shows that everything is depending on k (if we asume no latency for the network). That is, $k = \frac{target}{g+1}$, the number of tries each stakeholder has until the target time, is the most decisive factor. For low values of k the CDF is a bit higher , which is desirable. It is also more safe, as it restricts the stakers, making grind type attacks harder. Unfortunately, it also produces more orphaned blocks (see below), which have a negative impact in network efficiency and security.

Finally, we must not forget that we asumed no latency or propogation time. In real world the latency may be considerable and there is also the propogation time (time for the block to be "pushed"). That means that transmition of data is not instant. A research [4] showed that for bitcoin - a network with many nodes around all the globe - the median propogation time is 6.5 seconds, while the average time is 12.6 second. That is natural as some nodes will take longer than normal to respond (or even lose connection). In section 2.1 we made our own alalysis and tests of the propagation time. The reader should study this section to have a better idea about the network behavior of Ecochain.

$target$	g	$k = \frac{target}{(g+1)}$	p	p for $k = 1$	CDF for k	$p_{or}\%$
32 sec	7	4	0.25	0.25	0.6836	6.25
64 sec	7	8	0.125	0.125	0.6564	1.56
64 sec	31	2	0.5	0.5	0.75	25
128 sec	1	64	0.015625	0.015625	0.635	0.02
128 sec	3	32	0.03125	0.03125	0.6379	0.1
128 sec	1	64	0.015625	0.015625	0.635	0.02
256 sec	7	32	0.03125	0.03125	0.6379	0.1

Figure 1: probability distribution of block creation time

4 Network effectivness and security

What is the probabilty of the creation of at least one orphaned block? The probability for a node $n1$ to find one block in the next interval ("tick"), is

$$P(X = 1) = (1 - P)^{1-1}P = (1 - P)^0 P = 1 * P = P$$

, where P is the total propability of all stackholders. Asuming a great distribution of the coin (stakes) , the probability for a second stakeholder, lets says $n2$, to find another valid block in the same period of time (before the next interval) is around P also. That is because, as we have seen from above

$$P = p \sum_{1}^{s} b_i = p * B$$

Taken the fact that $n1$ with balance b_{n1} has found a valid block, the probability that another can be found is:

$$P' = p \sum_{1}^{n_1-1} b_i + p \sum_{n_1+1}^{s} b_i = p \sum_{1}^{n_1-1} b_i + p * b_{n1} + p \sum_{n_1+1}^{s} b_i - p * b_{n_1} =$$

$$= p \sum_{1}^{s} -p * b_{n_1} = p * B - p * b_{n_1} = P - p * b_{n_1} \approx P$$

since $\frac{b_{n_1}}{B} \approx 0$ (we have assumed large distribution to stakeholders).
In short, the probabilty P_{or} for an orphaned blocked to be formed will be

$$P_{or} = P * P' \approx P * P = P^2$$

Example: With a target of 64 sec of blockcreation and granularity $g = 7$ we have $k = \frac{target}{g+1} = \frac{64}{7+1} = 8$, so $P = \frac{1}{k} = \frac{1}{8}$ and $P_{or} \approx P^2 = \frac{1^2}{8^2} = \frac{1}{64} = 1.5625\%$ So the orphaned blocks aren't so many - only one orphaned block out of 64 will be created if $k = 8$. But what happens if $k = 4$. In this occation, $P_{or} \approx \frac{1}{k^2} = \frac{1}{4^2} = \frac{1}{16} = 6.25\%$

A question arises why we should care about the percentage of orphaned blocks. Orphaned blocks influence the efficiency of the network, which in turn has an impact on the security of the chain. But just for a memont let's asume that there aren't any ophaned blocks. How stackholders can succesfully attack the chain? In PoS if an entity or group has more than 50% of the coins he can launch a succesful attack, reversing blocks and rewritting history. It is not very difficult to see why this is happening. Let's say that the an attacker a has B_a balance in his account(s). He wants to reverse the previous n blocks, so he is "going back" and starts staking from the desired point. As the algorithm approves the longer chain his personal chain is not validated because he is n block back in history. But because he stakes only in his chain, now the rest of the stakeholders have not P probability but less than $\frac{P}{2}$ as the total available balance now is $B' = B - B_a$, where B is the total ammount of coins. Simply

$$B_a > \frac{B}{2} \Leftrightarrow p * B_a > p * \frac{B}{2} \Rightarrow p * B_a > p * B' \Leftrightarrow P_a > P_r$$

where P_a and P_r is the probability to form a block for attacker and the rest of the network respectively. As time passes the attacker will form new blocks more often than the rest of the network until his chain finally passes in length the original chain length. At this point the consensus algorithm will consider his chain as valid and the original chain as invalid. So he succesfully reversed (rewritten) n blocks.

This kind of attack, which in fact is also possible in PoW consensus protocol, can be even easier if there are orphaned blocks. Let us define the efficiency of the network as a function $ef(o) = 1 - o$, where o is the percentage of orphaned blocks. The limit for the previous attack to take place will be

$$s(ef) = \frac{ef}{1 + ef}$$

where s is safety function and ef the efficiency function. For example , in case of absense of orphaned blocks we have

$$o = 0, ef(o) = 1 - o = 1 - 0 = 1$$

$$s(ef) = \frac{ef}{1 + ef} \Leftarrow s(1) = \frac{1}{1 + 1} \Leftrightarrow s(1) = 0.5$$

k	p	Orphaned %	net.ef. (ef)%	safety ("democracy" attack)%
64	0.015625	0.0244140625	99.9755859375	0.49993895739226
32	0.03125	0.09765625	99.90234375	0.499755740107474
16	0.0625	0.390625	99.609375	0.499021526418787
8	0.125	1.5625	98.4375	0.496062992125984
4	0.25	6.25	93.75	0.483870967741936
2	0.5	25	75	0.428571428571429

Table 1: orphaned blocks , network efficiency and safety

So without orphaned blocks an attacker needs more than 50% of the coin. But, for $o = 6.25\%$ we have $ef(0.0625) = 1 - 0.0625 = 0.9375 = 93.75\%$ and the safety limit is:

$$s(ef) = \frac{ef}{1+ef} = \frac{0.9375}{1+0.9375} \approx 0.4839 = 48.39\%$$

It is clear that the safety of the system is lower now.
In the above table we can see how orphaned blocks are related to the security of the chain.

5 Glossary

Consensus algorithm: The algorithm or protocol is the proccess of a distributed system to reach consensus. For blockchain, it means the process of how the nodes create the next block , validate and record transactions on the chain.
Geometric distribution: a type of dicrete probabilistic distribution.
Democracy attack (PoS): Attack where the majority of stakholders can reverse the history of the chain.
Stake grinding attack: A type of attach which the attacker use computational power to make a succesful fork; that is , it converts PoS to PoW computing many hashes in sidechains.
Proof of Stake: Consensus algorithm based on staking (capital resources).
Proof of Work: Consensus algorithm based on heavy computation (hardware and electricity resources).

References

[1] *Gossip protocol* `https://en.wikipedia.org/wiki/Gossip_protocol`

[2] *Nielsen's Law* `http://wiki.p2pfoundation.net/Nielsen's_Law`

[3] *Kryder's Law* `https://en.wikipedia.org/wiki/Mark_Kryder#Kryder's_law_projection`

[4] *13-th IEEE International Conference on Peer-to-Peer Computing Information Propagation in the Bitcoin Network* https://www.tik.ee.ethz.ch/file/49318d3f56c1d525aabf7fda78b23fc0/P2P2013_041.pdf

[5] *Christian Walck , Hand-book on STATISTICAL DISTRIBUTIONS for experimentalists* http://www.stat.rice.edu/~dobelman/textfiles/DistributionsHandbook.pdf