Catalyst Network: Security considerations

Authors: Dr. Pauline Bernat*, Joseph Kearney† and Francesca Sage-Ling‡

Abstract

The scope of this paper is to estimate the level of resilience against any attempt to tamper with the ledger state by a malicious entity (or group of entities) gaining control of a sufficient number of producer nodes responsible for generating ledger state updates under Catalyst consensus mechanism. Such attack is commonly referred to as a 51% attack in consensus protocols notably based on the Proof-of-Work algorithm as it relies on any entity controlling more than half the network resource (hashing power) allocated to the ledger state update production. This paper also explores the level of confidence in the production ledger state update for various sizes of the producer pool.

*pauline.bernat@atlascity.io
†joseph.kearney@atlascity.io
‡fran.sl@atlascity.io

1 Security Considerations around the Consensus Mechanism

Whenever financial value is stored on a distributed system, there will be greater incentive to attack the system in the attempt to take control of financial assets or simply disrupt the system to create or destroy existing assets. With no centralised entity to control access and check the validity of assets exchanged over the network, the security and integrity of the network and ledger is managed by the network peers. Consensus and underlying protocols of the network are designed to allow transactions to take place and be validated in a trust-less environment.

Catalyst consensus mechanism is collaborative rather than competitive. Its protocol described in [1] consists in several phases executed by the producers during a ledger cycle. In order for these to reach consensus, a certain level of message propagation among the producers need to be reached during each phase. This paper investigates the threshold levels required that will ensure a fair and secure consensus to be reached between producer nodes within a ledger cycle.

This paper also covers the prospect of a 51% attack against the network and how ratio between the number of workers and producers can be chosen to prevent a malicious entity from being enabled to control the network.

1.1 Selection of Worker and Producer Nodes

The primary attack of concern for all blockchains and DLT platforms is the subversion of their consensus protocol and is generally referred to as a 51% attack. Such an attack is made possible when an entity or group of entities collude to have enough influence on the network to produce a block or ledger state update with invalid transactions, in the attempt to alter the ledger integrity. Depending on the protocol, the influence can be in computing power or number of nodes and must exceed 50% of the relevant resource.

An attack could be performed for many reasons aside attempting to steal money from a network, including to discredit or shake trust in a network. A consequence of a successful attack would likely be to reduce token prices. Although there is no tangible proof of this, it could explain why 51% attacks are not too common. Nevertheless, it remains important to prevent and mitigate the risk of an attack as much as possible.

The probability of a 51% attack (P_{51}) typically depends on the algorithm used to produce a valid block or ledger update.When considering Proof-of-Work algorithm and derivatives, P_{51} can be expressed as a function of the hash rate of network nodes. As the consensus-based protocol on the Catalyst network does not rely on solving a cryptographic puzzle, the concept of hash rate of nodes involved in the ledger state update is not relevant to quantify the probability or the cost of an attack on Catalyst network. The number of nodes involved in the production of a ledger state update is however relevant, as explained in this section.

The probability of a successful 51% attack on Catalyst network implies that a malicious entity (or group of entities) succeeds in controlling more than half the producer nodes selected to produce the ledger state update during a ledger cycle, giving that entity the power to tamper with the ledger state. The probability P_{51} depends on the following parameters:

- N : the total number of nodes in the worker pool.

- P : the subset of producer nodes selected to perform work for one ledger cycle ($P \leq N$).
- O : the number of malicious nodes in the worker pool ($0 \leq O \leq N$). This is a total subset of malicious nodes colluding to perform an attack on the network.
- p : the number of malicious nodes in the subset P of producers. ($0 \leq p \leq P$).

An attack can be considered successful for any value $p \in [p_0, P]$ where $p_0 = P/2 + 1$ which is equivalent to $p > 50\%P$. When $P \approx N$, *i.e.* the number of producers selected during a ledger cycle is very close to the total number of nodes in the worker pool, the absence of a randomness element in the selection of P producers makes it easy to compute the probability of a successful attack on the network: $P_{51} \approx O/N$. A malicious entity would know exactly when an attack can successfully be performed, that is when $O > N/2$.

When $N \gg P$, P_{51} can there be expressed by the discrete sum:

$$P_{51} = \sum_{p=p_0}^{P} P_A(p) \tag{1}$$

where $P_A(p)$ represents the probability of having p malicious nodes in the set P. When the ratio between the total number of nodes N and the number of nodes P is large ($N > 20 \times P$) it can be expressed as follows:

$$P_A(p) = \frac{\overbrace{\binom{O}{p}}^{\text{A}}\overbrace{\binom{N-O}{P-p}}^{\text{B}}}{\underbrace{\binom{N}{P}}_{\text{C}}} \tag{2}$$

A represents the number of possible combinations for choosing p nodes from O malicious nodes. B represents the number of possible combinations for choosing good (non-malicious) nodes for the remaining $N-O$ nodes in the worker pool. Finally, C corresponds to the number of available combinations for choosing P nodes from the pool of N nodes.

In equation 2, $P_A(p)$ is the probability mass function of a hypergeometric distribution over the set of parameters $\{N, O, P\}$. Note that such expression is valid for $max(0, O + P - N) \leq p \leq min(O, P)$.

There are two main arguments behind having a large number of N nodes:

- To account for the fact that most nodes with sufficient resources may want to join the worker pool and receive tokens as reward for their contribution to the ledger state management
- To make it increasingly costly for any malicious entity to control more than half the nodes.

As explained in the paper dedicated to Catalyst consensus paper [1], prior to joining the worker pool, nodes are part of a worker queue. Nodes in the worker pool are granted a work pass valid for finite period time. As a result, a varying number of nodes leaves the worker pool at each ledger cycle. Although the size of the worker pool might be constant (N nodes), the selection of nodes actually forming the worker pool changes over time. The mechanism used to define a score for nodes in the work queue is designed to prevent malicious nodes from gaining control of a large fraction of worker nodes. Nevertheless, as we derive the probability P_{51} in this section, we must stress that the fraction O/N may

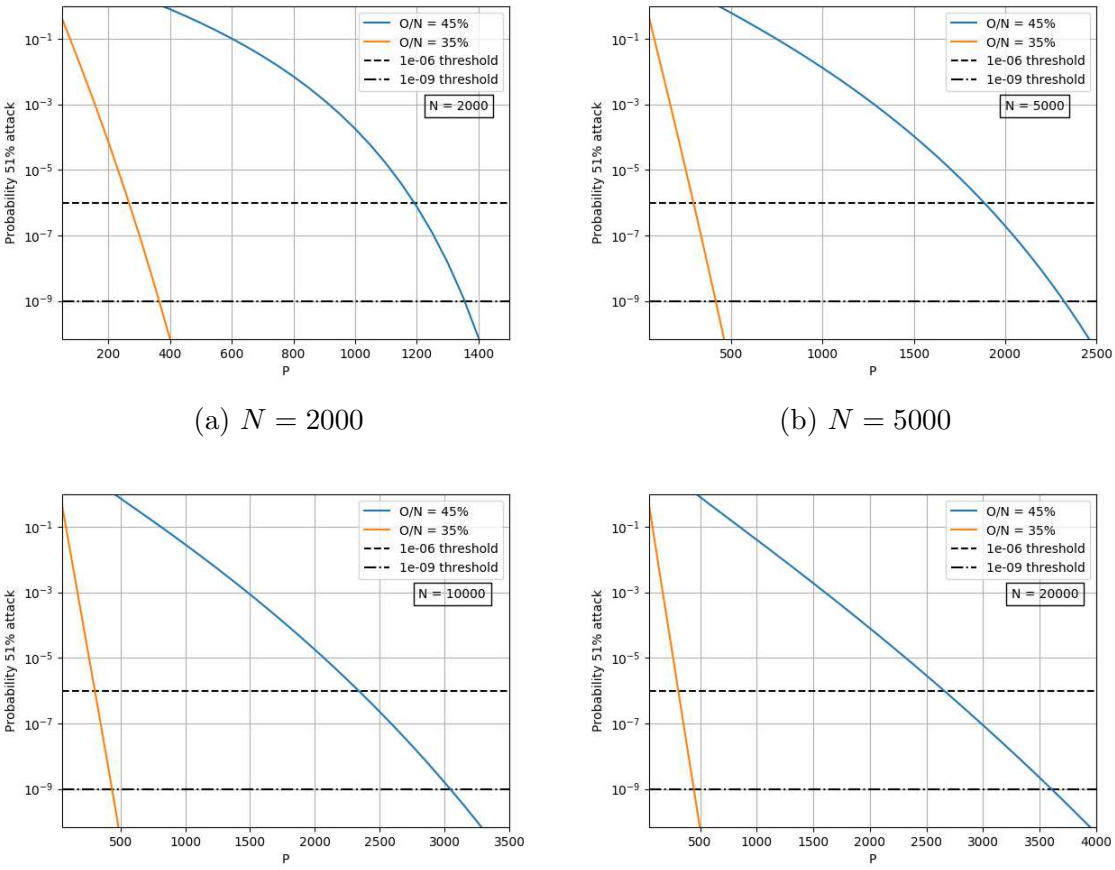
change (increase or decrease) over time and should be taken into account if computing the probability over a series of ledger cycles.

When $N \gg P$ the probability of a successful attack can be estimated using the cumulative hypergeometric distribution function (CDF) for $p \in [p_0, P]$. In this paper, we provide probability estimates obtained using *scipy.stats* Python library. The graphs presented are obtained using *matplotlib.pyplot* library. Rather than computing the CDF, the probability measurements are obtained using the survival probability (SDF), which is the inverse of CDF but is known to provide more accurate results[4].

As an example, let's assume a rather large number of nodes in the worker pool, $N = 20,000$, out of which 5% are selected as producers for a given cycle ($P = 1,000$). Let's further assume a ratio $O/N = 20\%$, *e.g.* 1 in every 5 nodes in the worker pool is controlled by a malicious entity ($O = 4000$). The probability of a successful attack is calculated using the SDF of an hypergeometric distribution using these set of parameters and amounts to: $P_{51} = 1 - SDF(20000, 4000, 1000) \approx 10^{-9}\%$. For the same set (N, P), the probability of a successful attack reaches 0.04% for $O/N = 45\%$ of malicious nodes in the worker pool.

Figure 2 shows the probability of a successful control of more than 50% of the producers as a function of the number of producers for four different worker pool sizes and two attack scenarios: when a malicious entity controls $O/N = 45\%$ of the worker nodes in blue, and in orange when a malicious controls $O/N = 35\%$ of the worker nodes in blue. For $N = 20000$, the probability remains below 10^{-9} if $P <\approx 4000$ while for a smaller worker pool size ($N = 5000$), the ratio P/N must be at close to 50% to prevent a successful control of more than 50% of the producers.

[4]See https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.stats.hypergeom.html for more details.

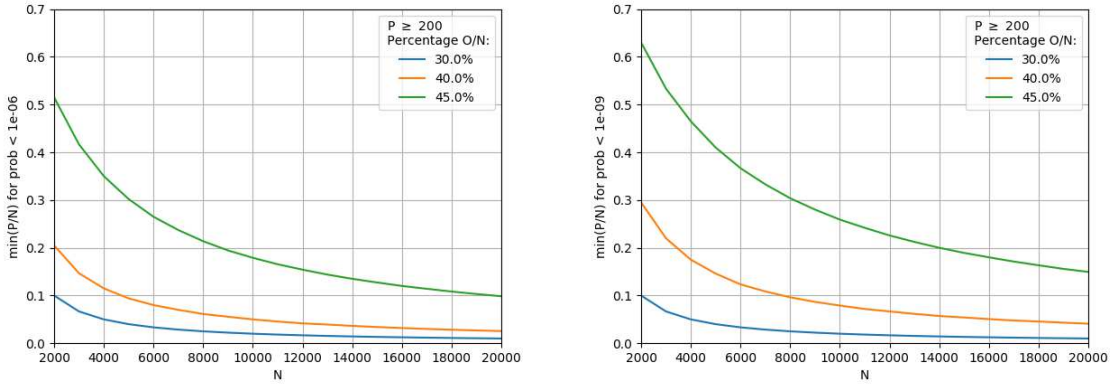


(a) $N = 2000$ (b) $N = 5000$

(c) $N = 10000$ (d) $N = 20000$

Figure 2: Probability of 51% attack as a function of P for various worker pool size ($N = \{2000, 5000, 10000, 20000\}$) when a malicious entity controls $O/N = 45\%$ of the worker nodes in blue, and in orange when a malicious controls $O/N = 35\%$

Figure 3 displays the minimum ratio P/N required to maintain a probability P_{51} below 10^{-6} and 10^{-9} for various malicious scenarios (O/N ratio between 30% and 45%) . This shows that as N increases the required P/N ratio required for the same security level decreases.



(a) $P_{51} < 10^{-6}$ (b) $P_{51} < 10^{-9}$

Figure 3: This graph shows the P/N ratio required for maintaining a probability of a 51% attack below two thresholds (10^{-6} on the left and 10^{-9} on the right) as a function of the number N of worker nodes.

This series of graphs gives a good indication on what pair of parameters (N, P) to

consider for a high resilience to 51% attack. Given a number of nodes in the worker pool, we can deduce the number of producer nodes to select during one ledger cycle. Inversely, given a number of producers for a ledger cycle, we can define a minimum size for the worker pool. As detailed in the next section, the number of producers selected for a ledger cycle is important to ensure that a consensus can be reached on the correct ledger state update to distribute to the rest of the network.

1.2 Production of a Ledger State Update

The previous section discusses the level of security against 51% attack when a malicious entity controlling more than half the producer nodes can attempt to tamper with the ledger state update. Specifically, the security of the consensus mechanism is considered as a function of the parameters (P, N). As N becomes large and the ratio P/N is low, it becomes very unlikely for a malicious entity to gain control of the worker pool, notwithstanding an increasingly expensive cost of attack.

In this section, we explore the confidence level associated with the production of a ledger state update. The quantities and thresholds defined during each phase of the ledger cycle are first enumerated. The values adapted or recommended for the thresholds are then discussed.

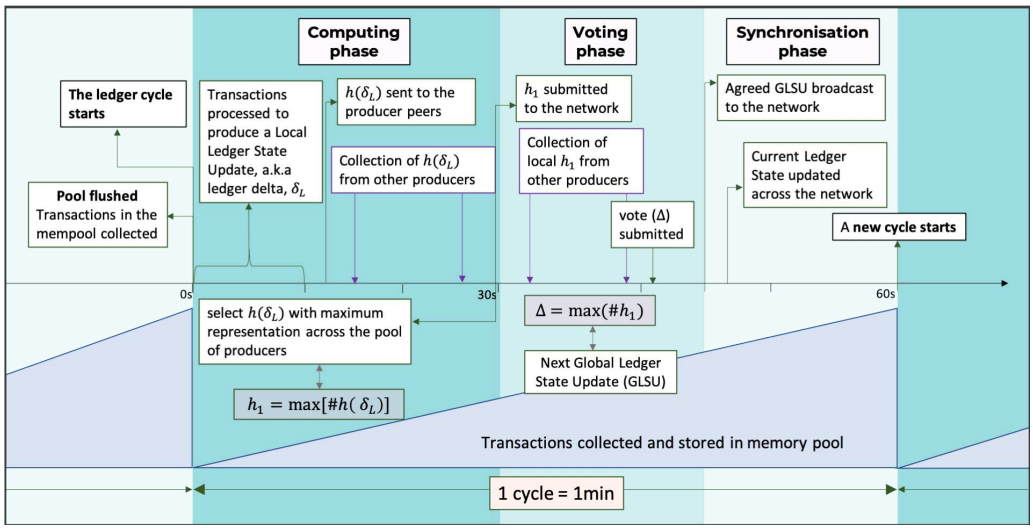


Figure 4: Illustration of the different phases followed by a producer during a ledger cycle.

A producer executes a series of steps in each phase of a ledger cycle, as illustrated in Figure 4. The producer can only move to a phase if a set of conditions are fulfilled in the previous phase. For a producer P_j, the first three phases consist of generating a quantity α_j that obeys certain criteria, and then broadcasting it to its producer peers while collecting the quantities α_k produced and broadcast by other producers $\{P_k\}_{k \in P/j}$.

1. **Construction phase:** $\alpha_j = h_j$
   h_j is the producer quantity generated by P_j, using the set of transactions stored in its mempool. It comprises the first hash value $h_{\Delta j}$, which includes the partial ledger state update (excluding any compensation entry) found by P_j and the compressed data structure for the transaction signatures, concatenated with P_j identifier Id_j: $h_j = h_{\Delta j} \mid\mid Id_j$.

   Participation All producers $\{P_j\}_{\forall j \in P}$ participate in the construction phase.

Time h_j must be broadcast before $t_p + \Delta t_{p0}$. Other producer quantities are collected during the time period $[t_p, t_p + \Delta t_p]$.

Quality Each transaction included in the ledger state update must verify a list of validity checks to ensure that the transaction being broadcast to the network had a valid structure and signature.

2. **Campaigning phase**: $\alpha_j = c_j$

   c_j is the producer candidate generated by P_j: $c_j = h_{\Delta j}^{maj} \mid\mid \#(L_j(prod)) \mid\mid Id_j$ with $h_{\Delta j}^{maj}$ the hash of the most common partial ledger state update found by P_j given the set of first hash values collected during the construction phase. $L_j(prod)$ is the partial list of identifiers compiled by P_j which includes the identifier of any producer having broadcast a first hash value corresponding to the most common, or candidate, partial ledger state update.The symbol # is used to represent a compressed data structure. As explained below we considered bloom filters to compress this list of identifiers.

   Participation All producers $\{P_j\}_{\forall j \in P}$ participate in the campaigning phase.

   Time c_j must be broadcast before $t_c + \Delta t_{c0}$. Other producer candidates are collected during the time period $[t_c, t_c + \Delta t_c]$.

   Quality
   - The number C_j of producer quantities collected by P_j must verify $C_j \geq C_{min}$.
   - The number of identical first hash values $C^{maj} = count[(h_{\Delta k} = h_{\Delta j}^{maj}) \; \forall \; k \in \{C_j\}]$ must verify $C^{maj} \geq C_{threshold}$.

3. **Voting phase**: $\alpha_j = v_j$

   v_j is the producer vote generated by P_j: $v_j = \mathcal{H}(LSU_j) \mid\mid \#(\mathcal{L}_j(vote)) \mid\mid Id_j$ which includes the hash (or second hash value) of the candidate ledger state update $LSU_j = L_E^f \mid\mid d_n \mid\mid L_{CE}$ generated by P_j. $L_j(vote)$ is the partial list of identifiers compiled by P_j which includes the identifier of any producer having broadcast a candidate partial ledger state update corresponding to the most common partial ledger state update. L_{CE} is the list of compensation entries created using the identifiers included in the complete and final list $L_n(prod)$ of C_n producers having broadcast a first hash value corresponding to the most common partial ledger state update. LSU_j thus includes the compensation entries for the producers $\{P_k\}_{\forall j \in C_n}$ who generated a producer quantity h_k verifying $h_{\Delta k} = \mathcal{H}(L_E^f \mid\mid d_n)$.

   Participation Only producers finding a $h^{maj} = max[unique(h_{\Delta k}^{maj}) \; \forall \; k \in \{V_j\}]$ satisfying $h^{maj} = h_j$ participate.

   Time v_j must be broadcast before $t_v + \Delta t_{v0}$. Other producer votes are collected during the time period $[t_v, t_v + \Delta t_v]$.

   Quality
   - The number V_j of producer candidates collected by P_j must verify $V_j \geq V_{min}$.
   - The number of identical partial ledger state update hashes $V^{maj} = count[(h_{\Delta k}^{maj} = h^{maj}) \; \forall \; k \in \{V_j\}]$ must verify $V^{maj} \geq V_{threshold}$.
   - $L_n(prod)$ includes the identifier of producers included in at least $P/2$ lists $\{\mathcal{L}_k(prod)\}_{k=1,..,V_j}$ associated to a producer candidate c_k satisfying $h_{\Delta k}^{maj} = h^{maj}$.

4. **Synchronisation phase**: $\alpha_j = o_j$

   o_j is the producer output generated and broadcast by P_j: $o_j = \mathcal{A}_n \mid\mid \#(\mathcal{L}_n(vote)) \mid\mid Id_j$. It includes the DFS content-based address $\mathcal{A}_n$ of the approved ledger state update ΔL_n.

   Participation All producers $\{P_j\}_{\forall j \in P}$ may participate in the synchronisation phase. However only the ones having successfully compiled the ledger state update $LSU_j = \Delta L_n$ may broadcast the address $\mathcal{A}_n$ to the network.

Time o_j must be broadcast before $t_s + \Delta t_{s0}$. User nodes must collect at least x identical addresses $\mathcal{A}_n$ during the time period $[t_s, t_s + \Delta t_s]$ and request the corresponding ledger state update to synchronise their local copy of the ledger.

Quality
- The number U_j of producer votes collected by P_j must verify $U_j \geq U_{min}$.
- The number of identical second hash values $U^{maj} = count[(\mathcal{H}(LSU_k) = \mathcal{H}(\Delta L_n)) \; \forall \; k \in \{U_j\}]$ must verify $U^{maj} > U_{threshold}$.
- $L_n(vote)$ includes the identifier of producers included in at least $C_n/2$ lists $\{\mathcal{L}_k(vote)\}_{k=1,..,C_n}$ associated to a vote v_k satisfying $\mathcal{H}(LSU_k) = \mathcal{H}(\Delta L_n)$. C_n corresponds to the number of identifiers of producers who correctly computed the partial ledger state update and are therefore included in $\mathcal{L}_n(prod)$.

The probability $\mathcal{P}(x > P/2)$ that $x > P/2$ during the synchronisation phase depends on a series of criteria:

1. $(C_{min}, C_{threshold})$: a producer needs to collect enough individual producer quantities (at least C_{min}) and find a majority (at least $C_{threshold}$) of identical partial ledger state update hashes to be able to issue a producer candidate. C_{min} is typically defined as a fraction of P: $C_{min} = f_C P$ with $0 < f_C < 1$. On the other hand, the definition of $C_{threshold}$ is more complex and depends on C_j.

   Although in theory $C_{threshold}$ could be set at $C_j/2$, a higher threshold must be chosen to allow a producer to decide on a candidate partial ledger state update in good confidence. Indeed, one must account for the statistical uncertainty associated to the ratio C^{maj}/C_j due to the size of the data sample used to compute this ratio. Moreover, there should be no ambiguity on the choice of a candidate, should for instance a second set of identical hash values of size close to $C_j/2$ be found in an attempt to tamper with the ledger state by a malicious entity controlling a large number of worker nodes. The confidence interval on a ratio $r = C^{maj}/C_j$ is defined as:

$$r \pm z\sqrt{\frac{r(1-r)}{C_j}} \tag{3}$$

   Where z is a score associated to the confidence level in r ($z = 4.22$ for a 99.999% confidence level) and the remaining expression is the standard error of the ratio estimate.

   In an scenario where only two types of first hash values are collected by a producer P_j, $x_1 = h^{maj}_{\Delta j}$ and $x_2 \neq h^{maj}_{\Delta j}$, P_j compiles the two ratios $r_1 = x_1/C_j$ and $r_2 = x_2/C_j$. Since $x_2 = C_j - x_1$, the two ratios have the same margin error: $\Delta r_1 = \Delta r_2$. As illustrated in Figure 5, if the margin error associated to the two ratios are such that $r_1 - \Delta r_1 < r_2 + \Delta r_2$, P_j cannot say with certainty that a majority of nodes agrees, even if $r_1 > 50\%$. A decision can only really be made if $r_1 > 50\% + \Delta r_1$. Figure 5(left) shows that for a $r_1 = 0.7$ the producer must collect at least $C_j = 110$ data in order to remove any ambiguity with a confidence level at 99.999%. Indeed, if $r_1 = 70\%$ and $C_j = 2000$, the second ratio r_2 would represent at best 30% of the data collected by the producer, the statistical uncertainty on these two ratios would leave a significant gap between 34.3% and 65.7%. For $V = 1000$, that gap would be reduced to [36.1%,63.9%], still large enough to give enough confidence to a producer that a clear majority of nodes agree on a common data. This is illustrated in Figure 5(right) when $R_1 = 0.6$. It can be seen that when $C_j = 200$ that there would be an overlap between the margin errors around r_1 and r_2, while when $C_j = 2000$ the producer can conclude with a confidence level of 99.999% that $r_1 > r_2$.

$C_{threshold}$ is therefore defined for confidence level (CL) as:

$$C_{threshold}(CL) = \left(0.5 + z(CL)\sqrt{\frac{C^{maj}(C_j - C^{maj})}{C_j^3}}\right) \times C_j \tag{4}$$

For a confidence level at 99.999%, $C_{threshold}$ can be expressed as:

$$C_{threshold}(99.999\%) = \left(0.5 + 4.22\sqrt{\frac{C^{maj}(C_j - C^{maj})}{C_j^3}}\right) \times C_j \tag{5}$$

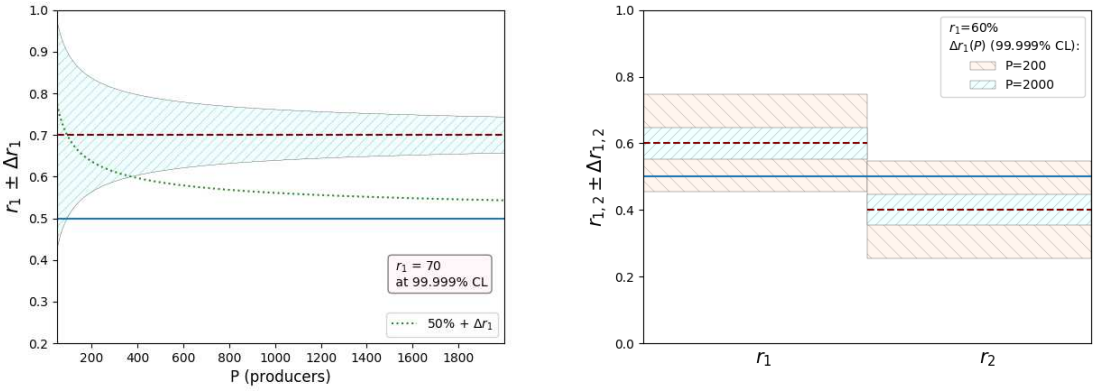


Figure 5: Left: $ri \pm \Delta r_i$ ($i = 1$ *or* 2) as a function of P, the size of the producers pool, when $r_1 = 60\%$. Right: $r \pm \Delta r$ at 99.999% confidence level, for two values of P (200, 2000) when only two types of hash are collected by a producer, when $r_1 = 70\%$.

2. $(V_{min}, V_{threshold})$: a producer needs to collect enough individual producer candidates (at least V_{min}) and find a majority (at least $V_{threshold}$) of identical partial ledger state updates embedded in the producer candidates to be able to issue a vote. V_{min} is typically defined as a fraction of P: $V_{min} = f_V P$ with $0 < f_V < 1$. $V_{threshold}$ is defined following the same approach considered for $C_{threshold}$:

$$V_{threshold}(99.999\%) = \left(0.5 + 4.22\sqrt{\frac{V^{maj}(V_j - V^{maj})}{V_j^3}}\right) \times V_j \tag{6}$$

3. $(U_{min}, U_{threshold}, C_n)$: a producer needs to collect enough individual producer votes (at least U_{min}), find a majority (at least $U_{threshold}$) of votes with identical second hash values, and hold a local copy of the ledger state update corresponding to the most common second hash value to be able to generate a producer output including the content-based address of the next ledger state update stored on DFS and broadcast it across the network. Two producer outputs are considered identical if they include the same DFS address of a complete ledger state update and the same list $\mathcal{L}_n(vote)$. Two complete ledger state updates are therefore identical notably when using the same list $\mathcal{L}_n(prod)$ to create the compensation entries. The list $\mathcal{L}_n(prod)$ comprises the identifiers of the C_n producers that produced the most popular partial ledger state update (without compensation entries) during the construction phase. C_n is typically defined as a fraction of P: $C_n = f_{prod} P$ with $0 < f_{prod} < 1$. U_{min} is thus defined as a fraction of C_n: $U_{min} = f_U C_n$ with $0 < f_U < 1$. $U_{threshold}$ is defined following the same approach considered for $C_{threshold}$:

$$U_{threshold}(99.999\%) = \left(0.5 + 4.22\sqrt{\frac{U^{maj}(U_j - U^{maj})}{U_j^3}}\right) \times U_j \tag{7}$$

In summary the probability $\mathcal{P}(x > P/2)$ that $x > P/2$ can be expressed as a function of $P, C_{min}, C_n, V_{min}, U_{min}$:

$$\mathcal{P}(x > P/2) = f(f_{prod}, f_C, f_V, f_U) \begin{cases} f_{prod} = C_n/P, f_C = C_{min}/P \\ f_V = V_{min}/P, f_U = U_{min}/C_n \end{cases}$$

The tests conducted on the gossip protocol implemented on Catalyst suggest that a high percentage of nodes ($95-99\%$) in a large network ($\mathcal{O}(10,000)$ nodes) will successfully collect data from all their peers. Furthermore tests on a smaller network ($\mathcal{O}(1000)$ nodes) such as the sub-networks of workers and producers in charge of producing the ledger state update during a ledger cycle gives the percentage of nodes collecting the data from all their peers as close to $99 - 100\%$. As a result the numbers (C_j, V_j) of data collected by a producer P_j are naturally expected to be close to P and U_j close to C_n. A simulation analysis was done to determine the optimal common value of the parameters $(C_n, C_{min}, V_{min}, U_{min})$ to ensure a probability $\mathcal{P}(x > P/2)$ greater than 99.999% for various sizes of the producers pool P.

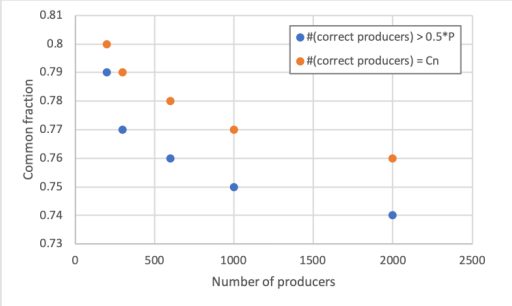


Figure 6: Minimum common set of parameters (f_C, f_V, f_U) as a function of P when $x \geq 0.5P$ (blue) and $x = C_n$ (orange).

Figure 6 displays the minimum common threshold found for $(f_{prod}, f_C, f_V, f_U)$ found for various size of pool of producers when $x \geq 0.5P$ and $x = C_n$. We observe that for $P = 200$, producers generate the correct ledger state update, $\mathcal{P}(x > P/2) > 99.999\%$ when all thresholds are set at 79%. The figure shows how the thresholds naturally decrease as the number P of nodes in the pool of producers increases. When $P \geq 1000$, a common threshold at 75% ensure than more than half the producers generate the same ledger state update. The mean number of producers found to generate the same value at $P = 1000$ when $f_{prod} = 75\%$ is 700, well above $0.5P$.

1.3 Bloom Filters

In the previous section we denoted $\#(L_n(prod)$ and $\#(L_n(vote)$ the compressed data structures representing the final list of producers having correctly completes the campaigning and voting phases. The proposed method to verify that the correct list of producers having participated in these two phases of the consensus mechanism is found by a majority of producers for a given cycle is through the use of Bloom Filters.

Bloom Filters are space-efficient data structures that allow fast comparisons between data sets as well as efficiently allowing a user to query membership of an item to a data set. The base structure for a bloom filter is a bit vector. Bit vectors are a vector of elements within which each element is a 0 or 1. By default in an empty bloom filter each of these elements in the bit vector are set to 0. As elements are added to the bloom

filter, a hash of the element to be added is created. The bit value which equals the hash value has its bit flipped to 1. The use of bloom filters allows us to compare the lists held by producers efficiently. Given a data set, due to the nature of hashing functions each individual element from the data set can be hashed to create a unique value. Through the use of bloom filters you can guarantee that there will be no false negatives when querying a data set however false positives are possible. When checking if an item is included in a bloom filter, the answer is "No for sure, or Yes maybe".The probability of a false positive is controlled by the size of the bloom filter and how many elements are in the bloom filter. Given a false positive ratio F_{prob} and a number of items stored in the bloom filter n the size of the bloom filter in bits is $-(n \times log(F_{prob})) \div log(2)^2)$. Thereby there is a direct correlation between the number of elements being added and the probability of a false positive being found in a bloom filter [2].

Using bloom filter, a producer can easily verify a list of identifiers with the guarantee that no identifier from a correct producer was excluded but with a non null probability that it includes the identifier of a producer that did not successfully completed a phase. As a result such producer would be able to claim some reward. Although not ideal, such scenario is not dramatic provided that the rate of false positive count is kept low, thus impacting relatively little the proportion of rewards distributed to the correct producers.

Figure 7(left) shows the false positive rate parameter chosen for a bloom filter as a function of the number of producers (or item stored in the bloom filters) when selecting 5 hashing function per item, in order to maintain the false positive count of producers below 0.5 per cycle. Figure 7(right) shows the corresponding bloom filter size. For $P = 2000$, a bloom filter of 3.3 kBytes is necessary and a false positive count of 0.4 is found, which correspond to roughly 0.02% of the total number of correct producers.

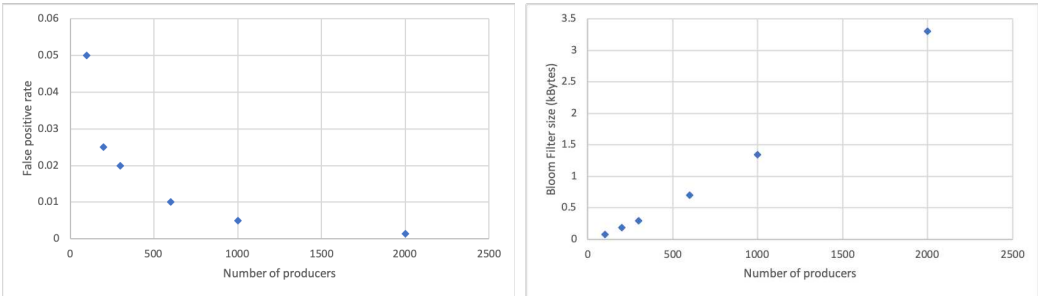


Figure 7: Left: false positive rate chosen for the bloom filter storing the list of producer identifiers. Right: size of the same bloom filter in kBytes to maintain a false positive count below 0.5 per ledgercycle. Minimum sets of parameters $(f_{prod}, f_C, f_V, f_U)$ found for $\mathcal{P}(x > P/2) > 99.999\%$ for P = 200 (left) and P = 500 (right).

2 Security Considerations around the Signature Scheme

Signatures for transactions on the Catalyst network are formed in a highly similar way regardless of whether the asset transfer embedded in said transaction is confidential or non-confidential. The signature scheme describes in the consensus paper [1] is a Schnorr-based signature scheme inspired from the Mu-Sig [3] signature scheme and presents similar vulnerabilities, as described below.

2.1 Rogue Key Attack

When Schnorr signatures are used to generate an aggregated signature of a transaction they are vulnerable to an attack known as Rogue Key attack. Rogue Key attacks performed by a malicious entity consists of generating an aggregated signature in such a way that they posses the public/private key pair for that signature. In the Schnorr signature scheme, the public key of participants are aggregated and the sum represent the public key associated to the signature. Assume that an honest participant use its public key Q_a in the transaction and a malicious participant possesses Q_b. By sending the public key $Q_m = Q_b - Q_a$ to the honest participant, the malicious entity have access to the transaction as they will hold the private key for Q_b. This is because when the keys are aggregated i.e. $Q_m + Q_b$ the aggregated signature would be Q_b, for which the malicious user holds the private key (the honest user would not). For an aggregated public key, there should be no user that has a private key equivalent as it should be used to create a signature that can be verified that all users in the transaction participated.

The aggregation of public keys used in Catalyst which is based on Mu-Sig [3] signature scheme that is not vulnerable to this form of attack. Mu-Sig is protected from this form of attack as the scheme does not require a user to demonstrate each public key, only the sum of all the public keys. By not verifying individual public keys, a key rogue attack is not possible. Only one public key is needed for the verification (the aggregated key) for which there will not be an equivalent private key.

2.2 Quantum Attack

Quantum computers pose a very real threat to the encryption techniques used in blockchains in the medium to long term [4]. The threat is through the use of Shor's algorithm. A quantum attacker using Shor's algorithm on a quantum computer can gain an exponential speed-up in solving the discrete logarithmic problem. The assumption of security the discrete logarithmic functions the the primary basis as to which all elliptic curve cryptography is based. This means that even the schema demonstrated here will be vulnerable to attack. The use of aggregated signatures would provide some resistance, however this resistance would be negligible.

It must be impressed that this is not an issue for the near term and thereby, these schema are highly secure and efficient currently. The most efficient algorithm for classical computers to solve the discrete logarithm problem is the Pollard's rho[5], this does not run in polynomial time. While Catalyst is not currently resistant to quantum attack, this is a challenge that will be faced by all major distributed ledgers over time.

References

[1] J. Kearney P. Bernat and F. Sage-Ling. *Catalyst Network Research: a new Consensus Protocol*. https://www.atlascity.io/. July 2019.

[2] Thomas Hurst. *Bloom Filter Calculator*. https://hur.st/bloomfilter/.

[3] G. Maxwell et al. *Simple Schnorr Multi-Signatures with Applications to Bitcoin*. Cryptology ePrint Archive, Report 2018/068. https://eprint.iacr.org/2018/068. 2018.

[4] D. Aggarwal et al. "Quantum attacks on Bitcoin, and how to protect against them. Quantum Physics". In: *arXiv:1710.10377* (Oct. 2017).

[5] A. Koundinya et al. “Performance Analysis of Parallel Pollard’s Rho Factoring Algorithm”. In: *arXiv:1305.4365* (May 2013).