Finality on the VECTOR blockchain

A TECHNICAL REPORT COMMISSIONED BY THE APEX FUSION FOUNDATION

Peter Thompson, Neil Davies
Predictable Network Solutions Ltd.

Duncan Coutts
Well-Typed LLP

August 2025, v1.0

Abstract

The VECTOR blockchain
- has 10× the throughput of Cardano mainnet;
- given reasonable assumptions, 10 seconds after observing a transaction in a block it provides 99% confidence in finality (or 99.9% after 14s); and
- given more aggressive assumptions, after successfully *submitting* a transaction it provides *immediate* 98.6% confidence in eventual finality.

Contents

1 Introduction

This report is about the timing and probability of transaction finality in VECTOR. We look at four scenarios, each with their assumptions, and for each scenario we present the timing of finality that can be expected.

This report should serve as a tool for DApp authors: based on an understanding of their use case, DApp authors can decide which scenario(s) are appropriate (noting that different assumptions may make sense for different parts of the same application) and use this report to establish what timing of finality they can expect.

The approach that we take is to develop probabilistic models of the scenarios in the ΔQ probability calculus[Haeri et al., 2022]. These models are calibrated using benchmark data from VECTOR testnets[1] (see Section 3.4). The models can answer precise questions of timing for different confidence levels. In the latter sections of this report we will justify why we believe these are appropriate models, but in short it is based on our collective experience (since 2017) of implementing, operating and analysing Ouroboros Praos and the Cardano mainnet.

1.1 Asking the right finality question

A transaction is final when it is permanent: irreversibly incorporated into the blockchain ledger.

In a consensus protocol like Ouroboros[David et al., 2018] in Cardano or Nakamoto consensus in Bitcoin, finality is probabilistic: as the protocol proceeds, there is a (rapidly diminishing) chance that blocks – and thus the transactions within them – can be rolled back and an alternative fork chosen.

[1]This can be easily updated as new benchmark data becomes available, or to investigate hypothetical "what if" scenarios.

So we might start with a simple question like "when will my transaction be final?", but this does not have a useful answer. In Bitcoin for example the technically right but practically useless answer would be: shortly before the heat death of the universe when the last Bitcoin block is forged. That is of course because Bitcoin blocks (and thus the transactions within them) are never really final: they're just probably final, with – for all but the most recent blocks – a very high degree of probability.

So our question should be expressed in terms of some level of confidence. It is not quite as simple as asking "what is the probability that my transaction will be final?". This question does have an answer at any point in time, but the answer changes relatively rapidly during the period after a transaction is submitted or observed in a block. In Bitcoin, we would observe blocks being built on top of the block containing our transaction and after enough blocks we would conclude that the probability that our transaction is final is good enough for our purposes. By itself, waiting is not enough. Crucially, we have to observe that the block containing our transaction *remains in the blockchain* during the time we wait. The chance that our transaction did get rolled back while we waited is typically not of interest: what is of interest is the residual probability that the transaction might not end up on the chain and how long we had to wait to get to that point.

The timing of (probabilistic) finality is the time we have to wait – while observing our transaction remains on the chain – before we get to a point where the residual probability that the transaction might not end up on the chain falls below an acceptable level.

So this gets us to the final form of our question:

> "How long do I need to wait – while seeing nothing bad happen – *to be 99% confident* that my transaction will become final?"

Or similarly for any confidence level. This report aims to answer this question for VECTOR.

The answers – of course – depend on the assumptions and the level of confidence that is desired. Different assumptions are appropriate for different users or uses (e.g. different DApps). We aim to provide a menu of plausible assumptions that cover a range of use cases, to analyse the scenarios arising from these assumptions and give concrete answers to the finality question, for a range of confidence levels.

Readers may wonder why we have laboured for so long on the form of the question. Apart from the fact that it is somewhat subtle, we want to avoid confusion when we get to our final scenario, in which we get a somewhat surprising answer. Given the assumptions of this scenario, it turns out that at time zero we can be pretty confident that our transaction will become final. Thus for some confidence levels the answer of how long we have to wait – the finality time – is zero. This seems counter-intuitive, since we normally think of waiting to see evidence. But if our question is really about confidence that our transaction *will become* final, then that does not in principle depend on even seeing our transaction in a block, provided we can be justifiably confident that the transaction will end up in a block and will become final.

2 What makes VECTOR different

VECTOR uses the same source code as the Cardano mainnet, but with a different set of parameters. VECTOR is designed to have higher capacity in terms of the number of transactions per second, and to have a lower latency for transaction finality.

When it comes to the timing of finality, VECTOR differs from the Cardano mainnet in three important respects:

- VECTOR produces blocks more frequently;
- VECTOR diffuses blocks in less time; and
- VECTOR is federated.

2.1 More frequent blocks

The classical analysis of Ouroboros Praos finality tells us to wait for a certain number of confirmation blocks. The exact number depends on various factors including what level of confidence we want, the active slot coefficient, the fraction of stake controlled by the adversary, and the grinding power of the adversary. Crucially however, whatever number of blocks we need to wait for, if blocks are produced more frequently, then waiting for a certain number of them takes less time.

VECTOR produces blocks five times more quickly than the Cardano mainnet. So all else being equal, this alone would have us reach a target level of confidence in finality five times faster[2].

2.2 Diffusing faster

The reason VECTOR can produce blocks five times more frequently than the Cardano mainnet is because it can diffuse the blocks much faster too.

Geography The primary reason it can do this is because VECTOR is geographically small: all the nodes are in European data centres that are at most 30 milliseconds apart from each other[3]. By comparison, the intercontinental links that Cardano mainnet must use can take well over 100 milliseconds (especially once TCP effects are taken into account).

Topology A secondary reason is that VECTOR is federated with only five block producing nodes. Each block producing node is connected to one or more co-located relays. These relays are part of a set of relays which is small enough that it is practical to use a *fully connected* topology. This means that every block producer is no more than three hops from all other block producers, and only one 30ms link must be traversed (since the relays are co-located in the same data centre with block producers and so have very short communication time). By contrast, decentralised networks like Cardano mainnet or PRIME have many more nodes, more hops, and many much longer communication links.

It is therefore not surprising that VECTOR benchmarks show that most blocks are diffused in less than one second, as shown in Figure 1.

Diffusing blocks faster is a means to the end of producing blocks more frequently which improves finality times. It also reduces the chances for longer forks, and fewer forks also improves finality times. Diffusing most blocks within one second means that for VECTOR , most blocks are diffused within a single *slot*. If all blocks were always diffused within a single slot, there would never be forks longer than length one, except due to adversarial action.

2.3 Trust

VECTOR is federated, with five organisations acting as stake pool operators (SPOs) operating block producing nodes. These organisations have legal contracts surrounding their operations.

The intention of this permissioned, federated setup is that it should be reasonable to trust that these organisations are operating nodes that follow the protocol faithfully.

[2]It is slightly more subtle than this since VECTOR also uses a higher active slot coefficient, which also affects the number of confirmation blocks needed, but not to a significant degree.

[3]This is a conservative upper bound on the worst case.

If one can assume that all five block producers are acting honestly, then this also substantially changes the expected timing of finality. In the classical analysis of Ouroboros Praos, the fraction of stake controlled by the adversary is a crucial parameter that has a very significant effect on the number of blocks to wait for. The classical analysis does not consider the case of a 0% adversary[4], but it is at least bounded by a small adversary. By considering the zero adversary case specially, we can improve significantly on the general analysis.

3 A menu of assumptions

Not all assumptions are relevant, and not all combinations of assumptions lead to interesting different outcomes. We have tried to select combinations of assumptions that are useful, plausible in many applications and are interesting in that they lead to different outcomes.

3.1 Meaning of the assumptions

Before we look at the combinations of assumptions that lead to different scenarios, we review the assumptions that are involved. For DApp authors, understanding the meaning should help to evaluate if each assumption is true or reasonable in the context of the DApp.

Federated block producing nodes are not adversarial This assumption means that we trust the block producing nodes on VECTOR to follow the protocol faithfully. In normal Ouroboros Praos the "honest stake" assumption is that at least 50% of stake faithfully follows the protocol. This stronger assumption is that 100% of stake (used for block forging) faithfully follows the protocol.

The network is operating normally This assumption means that blocks are being transmitted between block producers in the normal amount of time. Technically this means that the long term behaviour of how long blocks take to make it across the network is a realistic prediction of future behaviour. There is an equivalent assumption in normal Ouroboros Praos, which is that blocks are *always* diffused between nodes within Δ time slots.

If we do *not* make this assumption, it is equivalent to assuming that large scale node and/or network failure can happen at any moment.

Races between agents to spend the same UTxO are not possible This means that for the transactions of interest, all of the transaction inputs are such that only one agent is authorised to spend them. This would be the case for example when spending from a single-user wallet: only the agent in control of the wallet's private keys can spend UTxOs from this wallet. It would not necessarily be the case for UTxOs in some kinds of script addresses: the script may allow different agents to race to spend the same UTxO. Whether this can happen or not is determined by the design of the application.

If we do *not* make this assumption, it is equivalent to assuming that such races between agents *are possible*.

System load This is an assumption about the relationship between demand and supply on the network as a whole. Supply is the capacity of the system to incorporate transactions, which is primarily determined by its maximum block size and the frequency of blocks. The demand is the number and size of transactions that all users collectively wish to submit to the system (in a given time frame). The specific assumption we need is that the demand follows a *Poisson distribution*:

1. there exists a average level of system demand relative to system supply, and

[4]Indeed some forms of this analysis would break down at 0

	Primary	Optimistic	Pessimistic	Suspicious
Honest stake	Yes: assume all honest stake	Yes: assume all honest stake	Yes: assume all honest stake	Yes: assume 80% honest stake
Network	Yes: assume operating normally	Yes: assume operating normally	No assumption	Yes: assume operating normally
Spend races	No assumption	Yes: assume no races	No assumption	No assumption
System load	No assumption	Yes: assume a level of load	No assumption	No assumption

Table 1: The assumptions for each scenario

2. arrival of each transaction is independent of the time since the previous one.

For the scenario that makes use of this assumption (see Section 4.2) the average level of system demand becomes a parameter of the model and so the results depend on what level of demand we assume.

If we do *not* make this assumption, it is equivalent to assuming that the demand can be arbitrary, which includes the case of it being far greater than the supply.

3.2 The scenarios

We define four scenarios, with different combinations of these assumptions:

1. the primary scenario (see Section 4.1);
2. the optimistic scenario (see Section 4.2);
3. the pessimistic scenario (see Section 4.3); and
4. the suspicious scenario (see Section 4.4).

The assumptions for each scenario are in Table 1.

3.3 Other assumptions and non-assumptions

While all of our scenarios have an assumption that most or all nodes are following the protocol faithfully, we do not assume that all block producing nodes are online and working at all times. If some nodes are slow or offline, the diffusion of blocks between the remaining nodes is not substantially impacted because blocks are diffused via other paths through the network.

On the other hand, if nodes are offline then the rate of block production is of course affected. This affects the time for a transaction to get into a block, but in a graceful rather than catastrophic way. We have not specifically modelled this scenario of one or more nodes being offline. It may be useful future work to incorporate this parameter into the models. We do not however expect substantial degradation from a minority of block producers being temporarily offline.

We do assume that our transaction is not near to its expiry time (or has no expiry time). Otherwise of course, if a transaction is submitted too close to its expiry time then minor delays translate into loss. This is not generally a difficult constraint for applications.

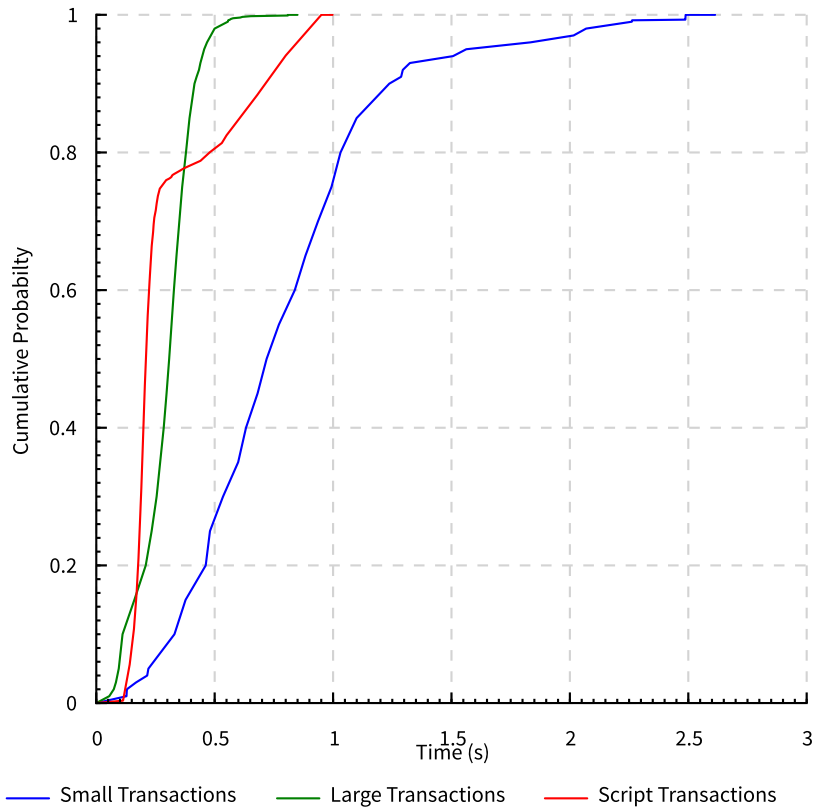


Figure 1: Probability of blocks reaching all nodes in a given time, for different transaction mixes

3.4 Benchmark data

Our models are calibrated using benchmark data from VECTOR testnets. Specifically we use measurements of the time to diffuse blocks across the network. This gives a ΔQ probability distribution of each block being diffused to all other nodes in a given time.

We used measurements from three different transaction mix scenarios, representing different extremes.

All small This consists of all transactions being small: 2 inputs and 2 outputs. This produces maximum sized blocks. This is the kind of transaction traditionally used to define a TPS measure. This mix is that one that can pack the maximum number of transactions into each block (since each one is so small).

All large This consists of all transactions being maximum size, by being full of metadata. Otherwise they also simply use 2 inputs and outputs. This produces full blocks but with far fewer transactions per block – only four since the maximum transaction size is a quarter of the block size.

All max scripts This consists of all transactions being scripts that use near to the maximum number of permitted script CPU units. Each transaction is however small in size in bytes. So this produces blocks that are not maximum size in bytes, but are (near) maximum in script CPU units.

The measurements are shown in Figure 1. We can see that the all-large and all-script transaction mixes give us blocks that can reliably be diffused within 1 second. On the other hand the all-small transaction mix has a longer tail to its distribution: blocks diffuse to all nodes in 1 second about 80% of the time, and about 3% of the time it takes between 2 and 3 seconds.

We generally make conservative assumptions for things that are hard to predict or control, and the transaction mix is one of these. Therefore all our models and their results in the rest of this report are based on the worst of these three diffusion distributions: the all-small mix. One caveat to this is that while the transaction mixes above do represent extremes, they are not the worst case extreme: that would be transactions with lots if inputs and outputs, no metadata, and scripts that maximise the CPU and memory units. This worst case extreme could be checked with further benchmark results.

It is worth noting that this is a baseline that could be improved upon. There are good reasons to think that with further benchmarking, profiling and optimisation that the long tail of the all-small distribution could be brought down to within 1 second (in the overwhelming majority of cases). This would provide improvements to the primary finality result in Section 4.1.

4 Headline results

In this section we present the main results, but not the detailed analysis or justification, which are in the remaining sections of this report.

4.1 The primary scenario

This scenario involves observing our transaction of interest in a block, and then waiting a short time until we are confident that there is no fork.

For this scenario we make the following assumptions:

- ✔ we assume that all block producers are honestly following the protocol;
- ✔ we assume the network is operating normally;
- ✖ no assumption about races to spend the same UTxO; and
- ✖ no assumption about system demand versus supply.

See Section 3.1 to review the meaning of each assumption.

4.1.1 Intuition

To get an intuition for this scenario, consider a simplified example. Suppose that by historical measurement of our network we know that blocks make it from one block producer to all other block producers within 5 seconds[5]. Suppose we observe a block with our transaction in it, and then there is a quiet period in which we observe no new blocks for a whole 10 seconds. If the network is still working, is it possible in this scenario for there to be any forks floating around in the network that are not yet resolved? No, it is not possible: 10 seconds is much more than enough time for all block producers and relays to have seen an alternative fork. Non-adversarial nodes will always select the better chain[6] and relay it to their peers. If they had seen a better chain then it would have been relayed to us within 5 seconds of us seeing the first one, so a 10 second gap is much more than enough. We can reasonably conclude that we are past the point where a fork could happen that would roll back our block. Thus our transaction is final.

[5]This is one of the Praos security assumptions for the Cardano mainnet: that $\Delta = 5$ slots.

[6]In the original description of Ouroboros Praos[David et al., 2018], chains of equal length are equally good and nodes will prefer to keep their existing chain. In Cardano's implementation of Praos however a tie breaker is used to order chains of equal length: a hash of the last block's VRF output. Thus while in the original Praos, chains have a partial order, in Cardano's implementation of Praos, chains have a total order.

4.1.2 No peace and quiet

It is also worth considering an example of what could go wrong if we do not observe a quiet period. Suppose we have two slot battles only 2 seconds apart: two nodes, A and B, are elected to produce a block in the same slot and suppose A includes our transaction but B does not, and then 2 seconds later both nodes A and B are both elected again to produce blocks. We have an observation node connected to a relay node R that is nearer to A than to B. What we observe at our node will closely follow what relay node R observes.

By our assumption for this example, it can take up to 5 seconds for these blocks to reach all other block producers, including nodes A and B. Since A and B produce their second blocks after 2 seconds which is less than the 5 seconds it takes to have seen the blocks from each other, they will both build on their own blocks and not extend each others. We assumed relay node R is close to node A, and thus blocks from A reach R before blocks from B. So R will see the first block from A (with our transaction in it), and then, before the first block from B has reached relay node R, it will see the second block from node A. At this point the longer chain is better and so R will adopt the chain with the two blocks from A, and ignore the block that arrives shortly thereafter from B. But at around the 8 second mark, when the second block from B arrives at R, then in the worst case (for us) the chain from B is now better than the chain from A and we will get a 2-block rollback. This rollback will drop our transaction from the chain.

Our observation node will see a filtered and delayed version of what relay node R observes. The delay is due to the network transmission delay between our observation node and the relay node R. We will see the first block from A at around t=5, the second block from A at around t=7 and very shortly thereafter the 2-block rollback and roll forward to the 2 blocks from B. Crucially, we did not observe a period of quiet of 5 seconds (5 slots) after the first block that contained our transaction.

4.1.3 Probabilities

In a real system it is not useful to talk about 100% of blocks being relayed within X seconds. Indeed, relying on extreme outliers does not provide a solid foundation. By the nature of outliers being rare it is hard to gather data on how extreme they are, and so it is hard to be confident that one has found a true upper bound. By contrast, instead of relying crucially on the behaviour of the top 0.1% of extreme outliers, if we can base our arguments on the bottom 99.9% then that is something we have a lot of data about and can be much more confident in.

It is useful to talk about 99%, or 99.9% of blocks being relayed within a certain time. Correspondingly, this lets us talk not of absolute finality, but a high confidence in finality. This block relaying time is something we can measure in synthetic benchmarks or in a real live system.

In VECTOR , the synthetic benchmarks show that blocks are relayed across the network rather quickly (as shown in Figure 1), so we can reach a high level of confidence in finality after observing quite a short period in which no new blocks arrive. Table 2 lists some points from the upper end of the distribution of block diffusion times on VECTOR , which are also the duration of quiet period we would want to observe before being confident that there will be no rollback. This gives us a welcome but initially surprising result that we would only have to wait just over two seconds without seeing any new blocks to be well over 99% confident that there will be no rollback.

4.1.4 Results

We have discussed that observing a quiet period will give us confidence in finality, but as we argued in Section 1.1, the question we want to answer is: how long do we *expect* to have to wait

Confidence desired	Milliseconds of quiet needed
95%	1,562 ms
99%	2,262 ms
99.9%	2,488 ms

Table 2: The duration of a quiet period we must observe after seeing a block to reach a level of confidence that there will be no rollback of the block, based on VECTOR testnet benchmarks.

Confidence desired	Expected number of slots to wait
95%	7
99%	10
99.9%	14

Table 3: The expected number of slots to wait (without seeing our block rolled back) to reach confidence that there will be no rollback of the block.

(without seeing a rollback) before we observe this period of quiet? This is different from the length of the quiet period itself.

The answer to this question comes from our model. Our model incorporates key factors including the distribution of block diffusion times and the probability of events like slot battles.

Table 3 gives the results, for a few selected confidence levels. As discussed in Section 4.1.3, we cannot go beyond confidence levels for which we have data from benchmarks. With current benchmarks we have enough sample data for the 99.9% confidence level but not beyond. Given more benchmark data, the model could give expectations for higher confidence levels.

4.1.5 Negative evidence

It is perhaps somewhat unsettling to rely on *negative* evidence for finality. It is certainly more traditional to rely on positive evidence. This negative evidence for finality relies crucially on the assumption that the network is operating normally, and in particular has not failed or encountered huge delays immediately after we observe the block with our transaction in it.

In particular one may worry that although the network between the relays is robust (since it has a high degree of redundancy), the link between our observation node and relays may not be so robust (if for example it only has one active link at once). If the "no network failure" assumption is too strong for a use case, then a reasonable step back would be to instead observe the next block arrival *after* a quiet period. This would provide evidence that the local network link has not failed, combined with the evidence of the quiet period.

4.2 The optimistic scenario

This scenario involves simply submitting our transaction: 'fire and forget'.

For this scenario we make the following assumptions:

- ✔ we assume that all block producers are honestly following the protocol;
- ✔ we assume the network is operating normally;
- ✔ we assume there can be no races to spend the same UTxO; and
- ✔ we make an assumption about the level of system demand versus supply.

See Section 3.1 to review the meaning of each assumption.

4.2.1 Intuition

The intuition for this scenario is that transaction submission and block production are in practice highly reliable: successfully submitting a transaction to one or more well-connected relays is highly likely to result in that transaction making it into a block.

This relies crucially on it *not* being possible for other agents to spend any of same transaction inputs as the transaction we want to submit. Otherwise, if it were possible, then whether our transaction makes it into a block first is simply a race between us and the other agent. This assumption also means that our transaction cannot be invalidated by other transactions, and therefore transaction order does not matter. Thus it does not matter which block or fork our transaction gets into, so long as it gets into one. So if we end up in scenarios where there are forks and rollbacks, then we will still be OK if our transaction ends up in another fork (since it cannot be invalidated by other transactions that get reordered ahead of it).

4.2.2 Surprising reliability

Cardano transaction submission was never required to be highly reliable. The design goal was that it should be mostly reliable, and that wallets should be responsible for transaction resubmission. In practice however transaction submission has turned out to be highly reliable. This is because transaction submission piggybacks on the general Cardano network layer, which is designed to be highly reliable by using a low trust approach that is resistant to adversarial behaviour at the network level. This approach was needed to ensure that block relaying is highly reliable, even in the presence of adversarial behaviour. Transactions can be relayed via any possible routes through the network graph[7], which has the effect that transaction submission is resilient to link or node delays or failures: transactions will still propagate to the subset of nodes that have not failed and are not experiencing long delays.

There is also strong empirical evidence that (properly configured) Cardano networks loose very few transactions. There is a Cardano stress-test benchmark[8], which is executed regularly to validate each new Cardano node release. In this benchmark, transaction generators are used to put the system into an overload situation and the generators submit different transactions to different relays – which is the situation most likely to lead to some transactions being lost. In the authors' experience supervising these benchmarks over many years – except when the system is misconfigured – these benchmarks loose very few transactions[9].

So what failures *could* occur that would result in our transaction not getting into all forks? It turns out that several unlikely things have to go wrong at once for this to occur – which explains why we do not see it in practice in the system benchmarks. We will describe the failure scenarios in Section 4.2.3 below.

For the theoretical analysis however, the complexity of these scenarios is too much to model. We only want to make numerical claims however where we have a reasonably sound theoretical basis. The approach that we take is to note that several things must go wrong, and just to model the simpler ones. This approach gives a very conservative over-approximation of the chance of

[7] The network graph is highly connected. Transactions are announced on all links, but typically transmitted only once to each node. So the transmission system has high redundancy but is also quite efficient.

[8] `https://github.com/IntersectMBO/cardano-node/tree/master/bench`

[9] Due to the way the transaction generators work, some loss is expected in the stress-test benchmarks, because it can create transactions that get rejected. In a voting-specific benchmark, the scripts check that no transactions carrying votes are lost.

failure. So while we will present the conservative numbers, we believe that the true probabilities are much better – but we have no way to quantify them.

4.2.3 The unlikely case of the lost transaction

The scenarios in which we can loose transactions involve the contents of the mempools being different on different nodes. This depends on the demand or load on the system as a whole, and hence the need for some assumption on the system load.

If the system is lightly loaded and our newly submitted transaction makes its way into the lower half[10] of the mempool on all nodes, then no matter which node or how many nodes next make a block (i.e. slot battles) then because the transaction will be in every fork it will be in the winning fork.

For things to go wrong, it needs to be the case that our transaction is in the lower half of the mempool on some block producing node(s) and in the upper half or not yet present in the mempool of other block producing nodes. This in itself is fairly common: the first can happen due to differing order of transaction arrival and the second can happen if there is sufficient system load that many relays have full mempools and so are not accepting new transactions yet. This gives us two scenarios.

1. In this scenario our transaction is in the mempool lower half on some block producing nodes and in the upper half on others. Now suppose a block is created from one of the nodes that has our transaction in the lower half, and thus the block contains our transaction. Suppose also that there is a slot battle or height battle and a competing block – which turns out to have the better VRF tie breaker – was from one of the nodes with the transaction in its mempool upper half, and thus the competing block does not contain our transaction. The least likely step is the following: suppose that the block with our transaction in it is relayed to *all or most* of the nodes that had our transaction in their mempool lower half *before* the competing block reaches them. This will cause those nodes to remove our transaction from their mempools. Then the competing (better) block arrives and is selected instead of the block with our transaction.

   This (unlikely) sequence of events has reduced the number of nodes that have our transaction in their mempools, but not eliminated it entirely. In particular – at a minimum – our transaction will remain in the mempool of the node that produced the competing block (which will eventually produce more blocks, and eventually one with our transaction of interest). It is now possible to hit this scenario again, which may reduce the number again. Eliminating our transaction entirely however requires hitting the second scenario, which is now possible since some nodes now do not have the transaction at all.

2. In this scenario our transaction is in the mempool lower half on some block producing nodes and not present at all on others. Now suppose a block is created from one of the nodes with our transaction in the lower half. And suppose there is a slot or height battle where the competing block – which turns out to have the better VRF tie breaker – comes from one of the nodes that did not have our transaction at all. The least likely step is the following: suppose the block with our transaction in it is relayed to *every* other node that does have the transaction in its mempool *before* the competing (better) block arrives at those same nodes. This includes intermediate relays, as well as block producers. In this case our transaction will be wiped from the mempool of every node that still had it, and the competing block without our transaction is selected instead.

[10] The mempool on a node is twice the size of a block, so for block producing nodes the lower and upper halves would form the content of the next two blocks made by that node.

This (very unlikely) sequence of events wipes out our transaction entirely. It has been lost.

We can also get a less extreme version where the transaction is only wiped out from some node's mempools. In which case we would need a repeat of this scenario to loose the transaction from the remaining nodes. A repeat is an independent chance and so again very unlikely.

We highlighted the least likely step in the sequence of events: one block reaching all of a certain category of node before a competing block. The reason this step is so unlikely is that there is no particular reason for the propagation of transactions between mempools and the propagation of the blocks to be significantly correlated. They do not originate from the same locations.

Note that to loose the transaction entirely we must encounter the second scenario, the first is not sufficient.

It is also worth noting that our transaction may still be in the mempool of the edge node that originally submitted to a relay, but unfortunately this does not help: each transaction is only submitted to each peer node once – even if at a later time the transaction *could* be resubmitted.

4.2.4 Demand vs Supply

We make no attempt to model these highly unlikely scenarios in full detail. In particular we do not model the network aspects. What we can do however is note that the failure case require at least one slot battle and requires encountering situations where mempools are full, which causes transactions not to be propagated promptly. We can model both of these.

For the latter we ask: 'what is the chance that we will encounter a full mempool for some assumed level of load?'. We expect to encounter full mempools more frequently the higher the demand is compared to the supply. We can model this precisely if the demand follows a Poisson distribution: a certain level of system demand relative to system supply, and (crucially) arrival of each transaction is independent of the time since the last one. The model is a 'bulk-service queue', and relies on an existing literature on queuing theory.

We combine the chance of encountering a full mempool with the chance of loosing a slot battle. This gives us our (very) conservative upper bound on the chance of loosing a transaction. The results in Table 4 below give the chance of *not* loosing the transaction, i.e. the chance the transaction will get into all forks and thus becoming final. The probability depends on the demand as a fraction of supply, since this is what determines the chance of encountering full mempools.

4.2.5 Immediate confidence of eventual finality

Offered load (%)	Minimum probability (%)
10	99.999
25	99.979
50	98.976
99	98.635
110	98.377
150	97.579

Table 4: Minimum expectation at point of submission of the transaction becoming final

If the assumptions for this scenario fit a use case, and in particular if is OK to measure or assume a level of demand on the system, then from the moment a relay accepts our transaction we can

have a high confidence that the transaction will make it into a block. We do not have to wait for anything else to reach a high confidence. Thus we get the somewhat surprising result that we achieve a high confidence of finality at time zero – measured from when our transaction is accepted into the mempool of a relay.

Table 4 gives a lower bound on the probability, given the assumed level of of demand. For example we see that when demand is 99% of supply, the probability of finality is (at least) 98.6%.

It would be misleading to summarise this without context as 'instant finality'. In particular instant finality might reasonably be interpreted to mean that *other* people or agents can see that our transaction will become final. That is not the case here since our transaction is not yet in any block. As we discussed in Section 1.1, the form of the question for timing of finality that we are analysing is a first-person perspective:

> "How long do I need to wait – while seeing nothing bad happen – to be 99% confident that my transaction will become final?"

And in this sense, given these assumptions, we can have a high confidence that a transaction we submit successfully will become final, and without having to wait any further.

4.2.6 The demand assumption

In theory the finality probability depends on the system demand, though as one can see from Table 4 the variation in probability is not hugely significant as the demand varies. If a bound on demand cannot simply be assumed, it can be measured: by observing how full the recent blocks are.

Observing recent demand cannot account for sudden demand spikes. Furthermore, if one application itself can generate massive short term demand, then the Poission assumptions break down. In this context, DApp authors should consider their own applications and other applications they co-habit with on the same chain.

In practice of course, empirical evidence suggests that the probability is much better than the theory can model, and so demand would be an insignificant factor. Furthermore, the simple mitigation of resubmitting lost transactions dramatically improves the probability of success (see Section 4.2.8 below).

4.2.7 Other failures

One special case of the scenarios described in Section 4.2.3 involves block producing nodes that re-join to the network (e.g. due to a process restart). For the first 10 seconds after re-joining any blocks they produce will be empty. This means they always act like a block producer without the transaction of interest, as in scenario 2. Node restarts of course happen independently of system demand, and their frequency is hard to model or make assumptions about. Nevertheless, the remaining steps in the scenario remain highly unlikely.

4.2.8 Failure mitigations

If the kind of probabilistic finality in this scenario is useful to enough applications – and crucially if the assumptions are reasonable for these applications – then it may be worth making improvements to the system to increase the probability of success.

There are a few approaches:

- Any wallet (including wallet functionality in DApps) should be able to re-submit transactions that do not make it onto the chain within a certain time frame. In VECTOR , if the

demand is even marginally less than supply then we expect submitted transactions to end up in – at worst – the fourth subsequent block. So wallets could be configured to resubmit the transaction if it is not seen within four blocks[11]. Resubmission dramatically increases the chances of eventual inclusion.

- The Cardano node could be modified so that when blocks are rolled back (as part of switching fork) the transactions in the blocks that are rolled back are re-inserted into the local mempool. This would avoid loosing those transactions when switching fork, which is the root cause of the failures in the scenarios in Section 4.2.3. In the long term this is likely to be the most reliable method.

  In this possible future, where transaction relaying becomes highly reliable (in theory as well as in practice), the greatest remaining risk of failure would likely be in the initial submission. Submitting to a single node is a single point of failure. It is common practice to submit to a single local node which is then connected to many public relays. This suggests a couple ideas:

  1. Use submission agents submit to multiple public relays and report the number that acknowledge accepting the transaction.
  2. Make the node a better submission agent itself: extend the local transaction submission protocol to allow reporting the number of peer relays that each transaction is accepted by. This is particularly appealing since the node already has the logic to maintain connections to public relays[12].

  In either approach, the idea is that once a transaction has been accepted by enough independent relays then it is highly likely to make it to all honest block producing nodes.

- Without modifying the Cardano node or wallets or using submission agents, an additional system agent could be used within the network to collect transactions and resubmit them when they get removed due to forks.

4.3 The pessimistic scenario

This scenario involves observing our transaction of interest in a block, and then waiting until we have seen additional blocks that build on top.

For this scenario we make the following assumptions:

- ✔ we assume that all block producers are honestly following the protocol;
- ✖ no assumption that the network is operating normally;
- ✖ no assumption about races to spend the same UTxO; and
- ✖ no assumption about system demand versus supply.

See Section 3.1 to review the meaning of each assumption.

Given our assumption that all block producers are honest, then once a strict majority of block producers[13] have 'endorsed' the transaction – either by creating the original block containing the transaction, or creating another block on top – then it is no longer possible for the original

[11] One caveat is that edge nodes will keep transactions in their mempool and not resubmit them to the relays they are connected to. One may wish to use dedicated submission agents that submit directly to relays.

[12] And for trust-less networks like Cardano mainnet or PRIME: avoid or recover from eclipse attacks

[13] Strictly speaking, for this simple analysis we need the additional assumption that the stake is relatively evenly distributed. A slightly more sophisticated analysis would be needed to account for the current actual stake distribution. This analysis is simple enough that it could be done online by applications to calculate the most accurate and up-to-date version of these results.

Blocks	Probability	Seconds	Blocks	Probability	Seconds
1	0%	0	9	99.74%	32
2	0%	4	10	99.90%	36
3	48%	8	11	99.96%	40
4	76.8%	12	12	99.983%	44
5	90.2%	16	13	99.993%	48
6	96.0%	20	14	99.997%	52
7	98.4%	24	15	99.9989%	56
8	99.3%	28	16	99.9996%	60

Table 5: Probability of observing guaranteed finality, given N blocks deep without rollback

block to be rolled back. Our transaction is therefore final. This is not probabilistic. It is absolute. It is also direct evidence that users and applications can observe.

What is probabilistic is how many blocks we expect to have to wait for (without seeing the original block rolled back), before we will observe a majority of block producers endorse the block. Once we have observed this, however, there is no more uncertainty.

In the VECTOR network with 5 federated block producers (with equal stake), a majority of block producers means 3 or more. Note that we are only guaranteed finality *if* we see 3 distinct block producers endorsing the transaction. It is also possible of course for blocks to be rolled back before we get to guaranteed finality.

Once we have seen this evidence, it is a guarantee that is resistant to failures such as network partitions, or individual nodes going offline or crashing. The only way a transaction could be lost at this stage is if *all* nodes experienced disk failure or there was a correlated software bug such that recent blocks are lost.

Table 5 shows the relationship between how many blocks we expect to have to wait for – without seeing our block rolled back – before we see the evidence that guarantees finality. This is for the VECTOR network with 5 federated block producers with equal stake. In the table, block 1 is the block containing our transaction of interest.

At the extreme end, if our transaction is still in the chain and is 16 blocks deep then we have a "five 9s" (99.999%) chance of having observed absolute finality. If there is no rollback during that time, we can expect that to happen in around one minute (60 seconds from the first block).

Applications targeting VECTOR specifically and that need a high degree of certainty, may wish to use this approach and wait to see blocks from 3 distinct block producers. This would however require extra tooling specific to VECTOR. Most off-the-shelf Cardano wallets do not support this directly: they support waiting a certain number of blocks. For these wallets, applications should simply pick a number of blocks to wait from Table 5 based on the level of confidence desired.

4.4 The suspicious scenario

For this scenario we make the following assumptions:

- ✔ we assume that at least 4 out of 5 (80% of) block producers are honestly following the protocol (though we don't know which ones);
- ✔ we assume the network is operating normally;
- ✖ no assumption about races to spend the same UTxO; and
- ✖ no assumption about system demand versus supply.

See Section 3.1 to review the meaning of each assumption.

The adversarial stake in this scenario is 20% (since the other 80% is honest). A general analysis of Ouroboros finality by Karpinski et al. [2021, see Table 2] gives the result that for a 20% adversary one should wait 13 blocks to get a 99.9% confidence of finality.

On VECTOR , one would expect to wait around 52 seconds before seeing the 13th confirmation block (after the original block with our transaction of interest).

One could apply this same style of analysis if we were to assume a 0% adversary: i.e. all federated block producers are honestly following the protocol. This style of analysis does not consider the zero adversary case, but the zero case is of course bounded by any minimum adversary. An earlier presentation of the same analysis [Kovalchuk et al., see Table 1] states that for a 5% adversary, it would be sufficient to wait 4 blocks to reach 99.9% confidence of finality.

On VECTOR , one would expect to wait around 16 seconds before seeing the 4th confirmation block (after the original block with our transaction of interest).

An important caveat for this analysis is that it does not take account of grinding, which is relevant when there is non-zero adversarial stake. The ability of adversaries to grind (taking advantage of lots of CPU power to compute their best possible moves) significantly increases the number of blocks that would be needed to achieve any given confidence threshold. To the best knowledge of the authors, analysis of Praos taking grinding into account has not yet been published in the literature[14].

A Ouroboros Praos Leader Selection

Ouroboros Praos has significant operational differences from typical consensus algorithms[15]. Instead of a slot leader schedule being pre-computed, each stakeholder separately computes its own schedule, based on its own private key. Since the overall schedule is the result of independent presudo-random computations, it is effectively a Poisson process. This creates the potential for both leadership clashes (where two or more stakeholders are scheduled to produce a block in the same slot, referred to as 'slot battles') and empty slots (where no stakeholder is scheduled to produce a block). In order to compensate for the empty slots, the slot time is kept short, so that the average rate of production of blocks is acceptable. The protocol is provably robust against message delays up to a parameter Δ (measured in slot-times), and its security degrades gracefully as the delays increase David et al. [2018].

This Poisson process has implications for the performance of the overall Cardano system:

1. More than one node can be elected in the same slot, producing a 'slot battle';
2. The non-uniform rate of production of blocks introduces a variable load on the block diffusion function;
3. Two or more nodes can become leaders in the same slot, which leads to a 'slot battle' in which only one of the generated blocks will be included in the chain;
4. The short slot time increases the probability that a block will not be fully diffused before the end of the slot (depending on the size of the block), and hence may not be available to a leader in the immediately following slot, causing a fork (referred to as a 'height battle');

[14]The authors are aware that such analysis has been done and would be happy to update this report to cite relevant published work.

[15]This section is based on Apfelmus et al. [2025].

5. Conversely, long sequences of empty slots (which must occur from time to time) allow all previous blocks to be diffused to every node, ensuring a consistent view of the chain to be established.

This introduces a set of trade-offs, determined by Praos parameters:

- Slot time;
- Slot frequency;
- Number of active nodes;
- Block size;
- Slot occupancy probability;
- Maximum diffusion delay Δ.

Which collectively affect the outcomes of the algorithm:

- Effective transaction rate;
- Wait time for inclusion in the chain;
- Probability of being included in a 'losing' fork (requiring transaction resubmission);
- Rate of growth of longest chain.

The parameters are summarised in Table 6.

Parameter	**Description**	**Notes**
N	Number of active nodes	In VECTOR this is 5
T_s	Duration of a slot	In VECTOR this is 1 second
f	Active slot fraction	$0 < f \leqslant 1$, In VECTOR this is $\frac{1}{4}$
Δ	Maximum number of slots before a diffused message is received	$\Delta \geqslant 1$

Table 6: Parameters for the Praos protocol

A.1 Distribution of leadership

From David et al. [2018], the probability of stakeholder U_i with relative stake α_i being leader in any slot is:

$$p_i = \Phi_f(\alpha_i) = 1 - (1 - f)^{\alpha_i}$$

If each of the N active nodes has an equal amount of stake, $\alpha_i = \frac{1}{N}$, and hence equal probability of being a leader in any particular slot, then we would have:

$$\forall_i \; p_i = 1 - (1 - f)^{\frac{1}{N}}$$

In general, the probability that stakeholder U_i is *not* the leader is $1 - p_i = (1 - f)^{\alpha_i}$, and so the probability that *no* stakeholder is the leader (i.e. we have an empty slot) is given by multiplying the probabilities (since each node decides independently whether it is the leader):

$$P_{\text{no leader}} = \prod_{i=1}^{N} (1 - f)^{\alpha_i} = (1 - f)^{\sum_{i=0}^{N} \alpha_i} = 1 - f$$

(Hence the definition of f as the active slot fraction). Note that this is independent of the actual distribution of stake.

Consequently, the probability of a run of m successive empty slots (since these are independent trials) is:

$$P_m^{NL} = P_{\text{m empty slots}} = (P_{\text{no leader}})^m = (1-f)^m$$

We can render this in Haskell as:

```
probNoLeader :: Rational   -- active slot fraction
             → Int         -- number of slots
             → Rational    -- probability of no leader
probNoLeader f m = (1 − f) ↑ m
```

More generally, when there are N nodes with an equal probability p of being a leader, the probability of m leaders in each slot is:

$$P_N^L(m) = \binom{N}{m} p^m (1-p)^{N-m}$$

where $p = 1 - (1-f)^{\frac{1}{N}}$ with N nodes having an equal distribution of stake.

This can be expressed in Haskell as:

```
probLeaders :: Rational   -- active slot fraction
            → Int         -- number of nodes
            → Int         -- number of leaders
            → Double      -- probability of m leaders
probLeaders f n m =
    fromIntegral (choose n m) ∗ p ↑ m ∗ (1 − p) ↑ (n − m)
  where
    p = 1 − (1 − fromRational f) ∗∗ (1 / fromIntegral n)
```

Specifically for VECTOR with $N = 5$ and $f = \frac{1}{4}$, the probability of m leaders in a slot is shown in Table 7.

Leaders	0	1	2	3	4	5
Probability	7.50e−1	2.22e−1	2.63e−2	1.56e−3	4.61e−5	5.46e−7

Table 7: Probability of m leaders in a slot in VECTOR

We can extend this to an analysis of slot battles: given that a block has been produced in a slot, the probability that there is a slot battle (i.e. that there are two or more leaders in the slot) is:

$$P^{SB} = 1 - (1-f)^{N-1}$$

since $(1-f)^{N-1}$ is the probability of there being *no* other leader in the slot (i.e. the other $N-1$ nodes are all not leaders).

The probability of *winning* a slot battle requires consideration of the number of other leaders. Since there is a 50/50 chance of winning against each of them, if there are m leaders in the slot, then the probability of winning is $\frac{1}{m}$. So the total probability of producing a winning block in any slot is:

$$\begin{aligned} P^{WSB} &= \sum_{m=1}^{N} \frac{P_N^L(m)}{m} \\ &= \sum_{m=1}^{N} \binom{N}{m} \frac{p^m (1-p)^{N-m}}{m} \end{aligned}$$

The probability of winning a slot battle given that a block has been produced is this probability divided by the probability of generating a block in the slot, i.e. P^{WSB}/f. This can be expressed in Haskell as:

```
probWinSlotBattle :: Rational   -- active slot fraction
                  → Int         -- number of nodes
                  → Double      -- probability of winning a slot battle
probWinSlotBattle f n =
    probWinningBlock / fromRational f
  where
    probWinningBlock = sum [ probLeaders f n m / fromIntegral m
                           | m ← [1 .. n]]
```

For VECTOR with $N = 5$ and $f = \frac{1}{4}$, the probability of winning a slot battle is 94.3%.

B Block Diffusion

Block diffusion is the process by which a block produced by a leader is propagated to other nodes in the network. This requires a series of steps:

1. The block forging node, having been elected, must construct the block;
2. Having constructed it, it must announce it to its neighbouring nodes;
3. The recipient node must determine that the block is novel and request it;
4. The block must be transferred;
5. The recipient node must validate the block, checking that it is well-formed and that it builds on the previous block.

To make the Cardano protocol robust against denial of service attacks, the block must be accepted as a valid extension to the node's current chain before it can be forwarded to other nodes.

The size of the block and the complexity of the scripts it contains determine the amount of time that these steps require, in particular the time taken to verify the block, and the time to transfer the block over the network.

To avoid redundant transmission of blocks, the protocol divides them into a small header and a larger body. The header contains the information necessary to establish that the block is one that has not been seen before, so the node can request the body containing the actual transactions. A node may receive the same header from multiple nodes, but it will only request the body from one of them. It can then forward the body to the other nodes that have requested it.

A minor compromise with regard to DoS resistance called 'Header Pipelining' (or 'Diffusion Pipelining') reduces the latency of block propagation in the absence of forks. In this approach, a new header is forwarded to the next node before the block body has been received, and the body is forwarded before it has been fully verified. To contain the risk of DoS attacks, the recipient node will not request another header from the sending node until the corresponding body has been received and verified, and every forwarding node must check:

1. That the header is correct before forwarding: i.e. the block correctly references its predecessor, and has been generated according to the Praos leadership schedule (verifiable-random-function (VRF) and block-signature validation);
2. That the block is complete before forwarding, i.e. the received (but not yet validated) body is indeed referenced by the header's body hash.

These ensure that any adversary can only inject malicious data to the extent that it controls stake.

Since we are interested in scenarios in which forks may occur, we will assume that the block diffusion process is in the non-pipelined mode, so that the body is fully received and validated before it is forwarded to other nodes. The full sequence of events (where node A is the block producer, node B is a directly connected node, and node Z is the next block producer) is then as follows:

1. Node A, having been elected, constructs a block;
2. It announces the new block to its neighbours (nodes B) by sending the header;
3. Node B validates the header and determines that it is novel (i.e. not already received from elsewhere);
4. Node B then Requests the block body;
5. The block is transferred from A to B;
6. Node B must check the block is complete;
7. Node B then Verifies the block body and adopts it;
8. Transfers the block to any neighbours that have requested it;
9. Node Z must adopt the block before constructing the next block.

Steps 2 to 7 are repeated for each node on the path from A to Z.

In the VECTOR network consisting of block producing nodes connected to fully-meshed relays, the path of each block is:

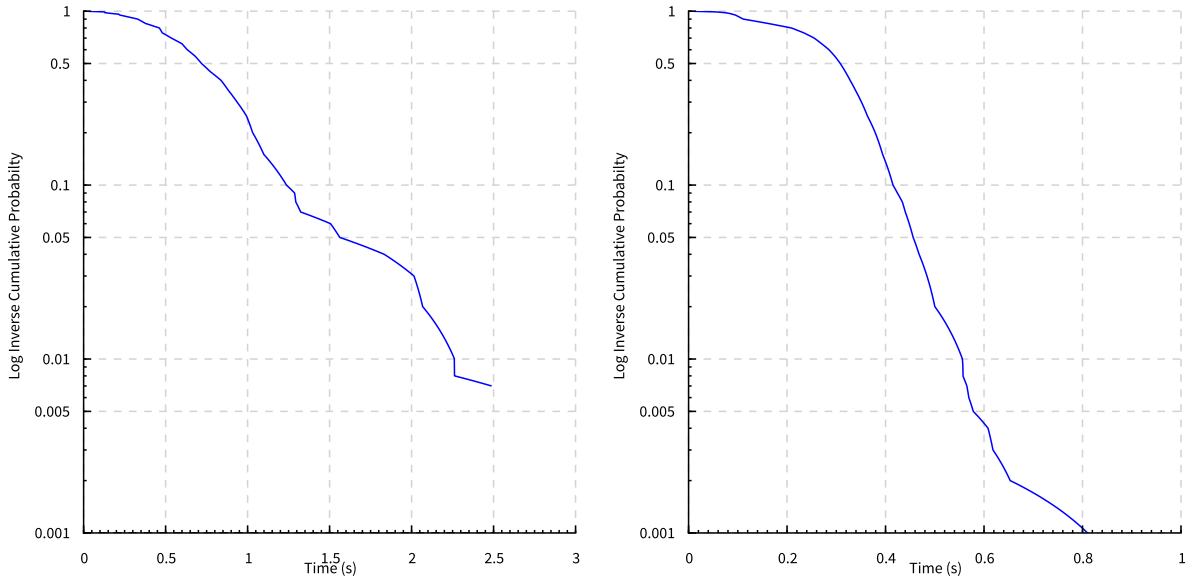
- The block producer node A produces a block;
- The block is transferred to the connected relay;
- The relay forwards the block to the other relays;
- Each relay forwards the block to the connected block producer nodes and any connected DApp clients;

The time taken to transmit a block is determined by the network latency, the speed of the network interface, the size of the block and any contention for network resources along the path. In VECTOR , all nodes are connected to a high-speed network, and each block producer is co-located with its relay, so the network latency of the first hop is very low. The relays are all located in Europe, so the latency between them is also low, typically on the order of a few tens of milliseconds. We will assume that a DApp client node is co-located with a relay, so the latency between the relay and the DApp client is also very low.

B.1 Block Diffusion Delay from Measurements

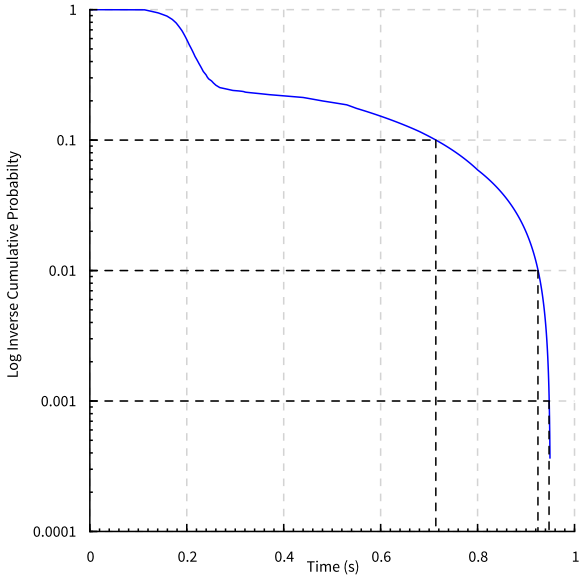
We can convert a list of measured diffusion delays to a ΔQ that can be used in the calculations above. The list of delays is a list of pairs of the form (p, d), where p is the percentage of transactions that are diffused within d seconds. The measured diffusion delays for blocks full of small, large and script transactions are shown in Figure 1. Inverse CDF plots for blocks with small, large and script transactions are shown in Figures 2a to 2c respectively.

For small transactions, diffusion succeeds in 1s with a probability of 75.9%; succeeds in 2s with a probability of 96.9%; and succeeds in 3s with a probability of 100.0%. For large transactions, diffusion succeeds in 1s with a probability of 100.0%.



(a) Blocks full of small transactions

(b) Blocks full of large transactions



(c) Blocks full of script transactions

Figure 2: Inverse CDFs of measured diffusion delays

C Interaction Between Block Diffusion and the Leader Selection Process

In any Cardano chain, a fork in the chain occurs whenever two or more nodes produce chains that extend the common prefix, but neither chain is an extension of the other. Assuming the nodes producing the blocks are honest and the code is correct, this can happen for two reasons:

- A block created in an earlier slot does not arrive in time to a node producing a block in a later slot;
- Two or more nodes are elected as a leader in the same slot.

The first case is a consequence of the block diffusion process, which is subject to delays and failures due to network conditions, node failures, and other factors. The second case is referred to as a 'slot battle', and is a consequence of the Poisson nature of the leader selection process. The probability of this was evaluated in Appendix A.1.

Following the framework described in Haeri et al. [2022] and Van Roy et al. [2022], we call of the time taken for a block to be diffused between two nodes the ΔQ ('delta Q') of the block diffusion outcome. Such a ΔQ can be calculated from first principles, or measured from a running system.

To calculate the probability of a fork following the generation of a block B at time 0, for each slot $n \geqslant 0$, we need to multiply:

- The probability that the block B has *not* been diffused in n slots, $1 - P_n^D$;
- The probability that no leader has been elected in the previous n slots, P_n^{NL}, and
- The probability that one or more leaders *are* elected in slot n, namely f.

where P_n^D is the probability of a block being transferred in n slots (the ΔQ of the block diffusion outcome), and P_n^{NL} is the probability of no leader being elected in n slots. Note that this includes the probability of a fork occurring in 0 slots (a 'slot battle'). So we can write:

$$P_{\text{fork}} = \sum_{n=0}^{\infty} (1 - P_n^D) \times P_n^{NL} \times f = f \sum_{n=0}^{\infty} (1 - P_n^D) \times (1 - f)^n$$

In code we can express this as:

```
forkProbability :: Rational   -- active slot fraction
                -> Rational   -- slot time
                -> DQ         -- transfer delay
                -> Rational
forkProbability f slotTime d =
  case deadline d of
    -- if the diffusion can fail, there's no obvious way to limit the sum
    Abandoned -> error "forkProbability: diffusion too uncertain"
    -- we can stop the sum when the diffusion probability reaches 1
    Occurs t  -> f * sum [ probNoLeader f i
                         * probNotDiffused i slotTime d
                         | i <- [0 .. ceiling (t / slotTime)] ]
  where
    -- probability that a block has not diffused in n slots
    probNotDiffused n s dq = 1 - successWithin dq (fromIntegral n * s)
```

where f is the active slot fraction, and d is the ΔQ for the diffusion delay. So, for instance, if we assume a diffusion delay uniformly distributed over 1 second, and an active slot fraction of 0.25, then the probability of a fork occurring is 0.25, i.e. only slot battles occur.

If we use the measured diffusion delay for small transactions and an active slot fraction of 0.25, then the probability of a fork occurring is (including both slot and height battles) 0.30, i.e. the probability of a fork occurring is quite low. Using the measured diffusion delay for large transactions and an active slot fraction of 0.25, the probability of a fork occurring is 0.25, i.e. only slot battles occur.

C.1 Forking and Transaction Finality

Having seen a transaction in a block, the question of finality is whether the transaction can be rolled back. From an observational point of view, the question is whether there can be a competing chain that we have not seen. To have a high degree of confidence that this is not the case, we need to wait until we have seen no blocks for a period of time that is longer than the maximum time it would take for a competing block to be diffused to us. Using the ΔQ of the diffusion delay, we can calculate the time to wait for the probability that an existing (competing) block has not diffused to the observer to be below a given threshold:

```
  -- how long to wait for the probability that an existing (competing) block
  -- has not diffused to the observer to be below a given threshold
probNoUnseenBlock :: DQ          -- the diffusion delay distribution
                  → Rational     -- the probability threshold
                  → Rational     -- the time to wait
probNoUnseenBlock d threshold =
  case quantile d threshold of
    Abandoned → error "probNoUnseenBlock: diffusion too uncertain"
    Occurs t  → t
```

However, we may want to know in advance how long we can expect to wait for the necessary gap in block production. This involves:

- The likelihood of a competing block being produced in the first place;
- The likelihood of a new block being added to our block before the competing block is diffused (this would cause nearby relays not to forward the competing block);
- The likelihood of a new block being simultaneously added to the competing chain (which prevents the competing chain from being discarded);
- and so on.

In each step there is at least one block producer who will *not* produce a block, since it is responsible for the competing chain, and so the probability of *no* competing block being produced is;

$$P_{\text{no. block N-1}} = \prod_{i=1}^{n-1} (1-f)^{\frac{1}{N}} = (1-f)^{\frac{N-1}{N}}$$

The probability that a competing block *is* produced is one minus this. Thus the probability of competing blocks being produced in n successive slots (since these are independent events) is:

$$P_{\text{competing block n}} = \prod_{i=1}^{n} 1 - P_{\text{no. block N-1}} = \prod_{i=1}^{n} 1 - (1-f)^{\frac{N-1}{N}} = (1-(1-f)^{\frac{N-1}{N}})^n$$

Slots	1	2	3	4	5	6
Probability	2.056e−1	4.226e−2	8.689e−3	1.786e−3	3.672e−4	7.549e−5

Table 8: Probability of competing blocks in VECTOR

For the VECTOR network with 5 nodes and an active slot fraction of 0.25, the probability of a competing block being produced in n slots is shown in Table 8. The number of slots we need to wait for this probability to be below a given threshold can be calculated as:

$$\begin{aligned} P_{\text{competing block n}} < d &\implies n \log(1-(1-f)^{\frac{N-1}{N}}) > \log(d) \\ &\implies n > \frac{\log(d)}{\log(1-(1-f)^{\frac{N-1}{N}})} \end{aligned}$$

Similarly, the number of slots to wait for the probability of a repeated slot battle to be below a given threshold is shown in Table 9.

Confidence	95%	99%	99.9%	99.99%	99.999%	99.9999%
Slots	2	3	5	6	8	9

Table 9: Number of slots to wait for a confidence of no competing blocks in VECTOR

C.2 System Quiescence

In order to have a high degree of confidence that there will be no rollback, there needs to be a period of time in which no new blocks are produced, long enough to achieve the required confidence threshold that all blocks produced have been diffused to all nodes.

The length of the quiet period depends on the ΔQ of the block diffusion outcome and the confidence threshold we want to achieve, as discussed in Appendix C. Converting this time into a number of slots gives the length of the quiet period we need. In Haskell this becomes

```
  -- how long a quiet period is needed for a given confidence
quietPeriod      :: DQ        -- the diffusion delay distribution
                 → Rational   -- slot time
                 → Rational   -- the confidence threshold
                 → Int        -- number of slots to wait
quietPeriod d s c =
  case quantile d c of
    Abandoned → error "quietPeriod: diffusion too uncertain"
    Occurs t  → ceiling (fromRational (t / s))
```

If we assume a diffusion delay distribution that is uniform between 0 and 1 with probability p, and otherwise takes 3 with probability $1-p$, the number of slots required for a quiet period is given in Table 10.

If we use the measured diffusion delay for blocks full of small, large or script transactions as described in Section 3.4, and an active slot fraction of 0.25, the number of slots required for a quiet period of a given length m with a given confidence c is shown in Table 11.

This leads to the question of how long we can expect to wait for such a period of quiet. As discussed previously, the probability of m successive slots being empty is given by $P_m^{NL} =$

Confidence	95%	99%	99.9%	99.99%	99.999%	99.9999%
p = 1.0	1	1	1	1	1	1
p = 0.99998	1	1	1	1	3	3
p = 0.9998	1	1	1	3	3	3
p = 0.998	1	1	3	3	3	3

Table 10: Number of slots to wait for a given confidence of quiescence

Confidence	95%	99%	99.9%	99.99%	99.999%
Blocks full of small Txs	3	3	3	3	3
Blocks full of large Txs	1	1	1	1	1
Blocks full of script Txs	1	1	1	1	1

Table 11: Number of slots to wait for a given confidence of quiescence in VECTOR

$(1-f)^m$. If we wait for $n > m$ slots, each of the first $n-m$ slots has could be the start of an empty run of m slots. Considering these as independent Bernoulli trials means that the probability of seeing no such runs is given by:

$$P_{n,m}^{\text{no empty run}} = \binom{n-m}{0}(1-(1-f)^m)^{n-m} = (1-(1-f)^m)^{n-m}$$

The probability of at least one quiet period of m slots in $n > m$ slots is then one minus this: $P_{n,m}^{NL} = 1-(1-(1-f)^m)^{n-m}$.

If we want a probability of at least c of this, then we can set $P_{n,m}^{NL} \geqslant c$ and solve for n.

$$\begin{aligned} P_{n,m}^{NL} \geqslant c &\implies 1-(1-(1-f)^m)^{n-m} \geqslant c \\ &\implies (1-(1-f)^m)^{n-m} \leqslant 1-c \\ &\implies n-m \geqslant \frac{\log(1-c)}{\log(1-(1-f)^m)} \\ &\implies n \geqslant m + \frac{\log(1-c)}{\log(1-(1-f)^m)} \end{aligned}$$

In Haskell this becomes

```
  -- how many slots to wait for a suitable quiet period with a given confidence
extraQuietPeriod :: Int                 -- required length of quiet period
                 → Probability DQ       -- the active slot fraction
                 → Rational             -- the confidence threshold
                 → Int                  -- number of slots to wait
extraQuietPeriod m f c =
  ceiling  ( log (fromRational (1 − c))
           / log (fromRational (1 − (1 − f) ↑ m)))
```

In VECTOR with an active slot fraction of 0.25, the number of slots to wait for a quiet period of a given length m with a given confidence c is given in Table 12.

If we combine the confidence of the required quiet period with the confidence that enough slots have passed to ensure that such a period has occurred, we can calculate the overall confidence that the transaction is finalised. The resulting number of slots to wait for a given confidence is given in Table 13.

Confidence	95 %	99%	99.9%	99.99%	99.999%	99.9999%
m = 1	3	4	5	7	9	10
m = 2	4	6	9	12	14	17
m = 3	6	9	13	17	22	26
m = 4	8	13	19	25	31	37

Table 12: Number of slots to wait for a probable quiesence interval in VECTOR

Confidence	95%	99%	99.9%	99.99%	99.999%	99.9999%
Blocks full of small Txs	7	10	14	19	23	27
Blocks full of large Txs	3	4	6	8	9	11
Blocks full of script Txs	3	4	6	8	9	11

Table 13: Number of slots to wait for a given confidence of quiescence in VECTOR

D Transaction Finality

The life cycle of a transaction in any Cardano chain is as follows:

1. The transaction is submitted to a node;
2. The node adds it to its mempool;
3. The transaction is then propagated to other nodes, which add it to their mempools;
4. The mempools become sufficiently empty that the transaction is in the lower part of the mempool;
5. The next node selected to build a block includes the transaction in its block;
6. The block is diffused to other nodes;
7. The block is adopted by the nodes, and the transaction is considered to be included in the chain;
8. Further blocks are produced, building on the block that included the transaction;
9. The transaction is considered to be finalised when a rollback of the chain is no longer possible.

It is important to note that a rollback can only occur if there is a fork in the chain, so once a transaction is included in a block for which there are no competing blocks of the same height, it is considered to be finalised. In an adversarial setting, there is the risk that an adversary could secretly construct an alternative chain and release it at some point in the future, causing a rollback if this chain is longer than the honest chain. The security of the Praos protocol derives from the the difficulty of constructing such a chain, which depends on the amount of stake held by the adversary, and the amount of time and computational power available to make it. In a non-adversarial setting, such as VECTOR , the risk of a rollback is much lower, since the nodes are trusted and the network is not open to adversarial nodes. Forks can only occur due to slot/height battles, the probability of which was discussed in Appendix C.

D.1 Expectation of finality at transaction submission

As was derived at he end of Appendix A.1, in VECTOR the probablity of winning at slot battle, ignoring any height battles, is 94.31%.

Some portion of the remaining probablity of 5.69% can still contribute to the successful inclusion of the transaction into the final chain, depending on the consistency of the mempool in the block generator that created the block that had the leading VRF.

In this residual case the final adopted block is produced by some other block producer (the one with the 'best' VRF). We make the assumption that the transaction of interest was in the mempool the said producer.

If the transaction was in the lower half of the mempool then the winning block, with regard to the transaction, contributes to the success criterion, if the transaction was in the upper half then the discussion is slightly more nuanced.

Noting that the winning block producer *does not* perform any rollbacks. The transaction of interest is still in that producers mempool and, when it becomes a slot leader, that transaction (now in the lower half of the mempool) will become the selected block with the 5.69% probability. Thus

$$P_{\text{tx final}} \geq 0.9431 + (1 - 0.9431) * (P_{\text{in lower}} + 0.9431 * P_{\text{in upper}}) \tag{1}$$

It is possible to estimate $P_{\text{in lower}}$ and $P_{\text{in upper}}$ by modelling the the mempool as a bulk-service queue to yield steady state probabilities for given loading and transaction mixes. They would give very pessimistic probabilities as they assume statistical independence, in the general scenario we are considering here the transaction is already assumed to have been in the lower half of the mempool of the block producer. Thus we are still calculating pessimistic bounds.

In Table 16 it can be seen that probabilities are only marginally effected by the transaction mix but are dominated by the offered load.

In choosing $P_{\text{in lower}}$ and $P_{\text{in upper}}$ for a given offered load, the most pessimistic values have been chosen from Table 16 - that being the ones with largest P_{blocked}.

Offered load (%)	$P_{\text{in lower}}$	$P_{\text{in upped}}$	Minimum probability (%)
10	$9.986 \cdot 10^{-1}$	$1.389 \cdot 10^{-3}$	99.999
25	$9.658 \cdot 10^{-1}$	$3.242 \cdot 10^{-2}$	99.979
50	$8.051 \cdot 10^{-1}$	$1.587 \cdot 10^{-2}$	98.976
99	$4.457 \cdot 10^{-1}$	$3.334 \cdot 10^{-1}$	98.635
110	$3.879 \cdot 10^{-1}$	$3.466 \cdot 10^{-1}$	98.377
150	$2.419 \cdot 10^{-1}$	$3.527 \cdot 10^{-1}$	97.579

Table 14: Minimum expectation of transaction becoming final at the point of submission (without considering height battles)

Some caveats, the calculation is based on the transaction having appeared in a produced block. The transaction submission protocol, that enforces inter-node, in-order, delivery and fair servicing of demand during overload (captured in P_{blocked}) ensures, almost certainly[16], that a submitted transaction[17] will eventually arrive in the mempool of a block producer.

Also, the mathematics underlying Table 16 makes assumptions on the arrival pattern of transactions being Poisson, at high loads (substantially over 100%) this assumption is less justifiable as backpressure in the transaction submission dominates the transaction submission behaviour.

[16]There is, in principle, a sequence of slot/height battles and rollbacks that might cause transactions to be erased from all mempools. The conditions for this involve a ...

[17]provided it remains valid, i.e. does not exceed its time-to-live during transit

D.2 Considering height battles

Consider the situation where the only fork lengths possible is precisely 1. They exist when all the produced blocks have the same common ancestor – one removed. In this case, in VECTOR , it implies that producer nodes have, at most, produced one block in the particular total diffusion time. The results in Table 7 holds. Hence the calculations above that result in Table 14 also holds.

Now consider forks of length 2, the following has to hold:

- The probability of block diffusing exceeding a slot duration must be non-zero,
- there must be two or more block producers in this time slot,
- there must have been two or more block producers in the preceeding slot, and
- the block producers in this slot must not all choose the same block from the proceeding slot.

Why these set of conditions? Ambiguity has to exist. Any one of them not holding means that the necessary ambiguity is not present to create a fork of length 2.

Assuming the first point a single active block producer in the second slot would either build on the block it had seen from the previous slot, uniquely, or would not have seen the block produced in the previous slot (due to diffusion) and hence would be part of the previous (fork length = 1) scenario. The second and third conditions are inherently staistically independent, the last point is difficult to quantify.

The probabilty of there being two or more slot leaders is 2.79%, so the probability of the necessary ambiguity to exist is $P_{\text{diverse}} * 7.79\text{e}{-4}$, where P_{diverse} represents the qualtification of the fourth bullet item above.

Forks of length 3 (which are maximum length theoretically possible in VECTOR given that all diffusion completes within 3 s) require that there be at least two block producers in all of the three slots (2.17e−5).

Table 15 captures the general relationships, not considering any potential successes in the residual values and worst case value for P_{diverse}. The table should be interpreted from last row upwards. In that row we consider the case of forks of length 3 (which we don't have the model to process, so consider them non-success) and lump the complement of that value into the previous case (forks of length 2) and so on.

N	Probability of diffusion occuring in this slot	Probability fork size 0	Probability fork size 1	Probability fork size 2	Probability fork size 3
1	0.759	9.43e−1	(1 - 9.43e−1)	N/A	N/A
2	0.210	$\geq$ (1 - 7.79e−4)		$\leq$ 7.79e−4	N/A
3	0.031	$\geq$ (1 - 2.17e−5)			$\leq$ 2.17e−5

Table 15: Interaction forking with diffusion probablities

As can be seen the probabilities of the "different from line above" circumstances is low (due to the conditions needed to arm the forking hazard). The consequence of this being that although probability of 24.1% of diffusion taking more than 1 second, the probability that the resulting forking scenario results in a fork longer than length 1 is $\leq$ 1.64e−2%. This is a very small change in the probabilities assumed in the no height battle case, (see Table 14) with the differences occuring only in the 4^{th} significant figure.

txs per block	mempool size (txs)	offered load (%)	mean mempool fill (blocks)	$P_{\text{in lower}}$	$P_{\text{in upper}}$	P_{blocked}
5	11	10	0.10	$9.986 \cdot 10^{-1}$	$1.389 \cdot 10^{-3}$	$5.740 \cdot 10^{-6}$
10	21	10	0.10	$9.995 \cdot 10^{-1}$	$4.904 \cdot 10^{-4}$	$4.794 \cdot 10^{-7}$
150	301	10	0.10	$9.999 \cdot 10^{-1}$	$5.860 \cdot 10^{-5}$	$3.661 \cdot 10^{-9}$
300	601	10	0.10	$9.999 \cdot 10^{-1}$	$5.174 \cdot 10^{-5}$	$2.765 \cdot 10^{-9}$
5	11	25	0.26	$9.658 \cdot 10^{-1}$	$3.242 \cdot 10^{-2}$	$1.812 \cdot 10^{-3}$
10	21	25	0.26	$9.723 \cdot 10^{-1}$	$2.675 \cdot 10^{-2}$	$9.578 \cdot 10^{-4}$
150	301	25	0.26	$9.797 \cdot 10^{-1}$	$1.995 \cdot 10^{-2}$	$3.926 \cdot 10^{-4}$
300	601	25	0.25	$9.799 \cdot 10^{-1}$	$1.968 \cdot 10^{-2}$	$3.771 \cdot 10^{-4}$
5	11	50	0.60	$8.051 \cdot 10^{-1}$	$1.587 \cdot 10^{-1}$	$3.625 \cdot 10^{-2}$
10	21	50	0.58	$8.088 \cdot 10^{-1}$	$1.603 \cdot 10^{-1}$	$3.088 \cdot 10^{-2}$
150	301	50	0.57	$8.140 \cdot 10^{-1}$	$1.605 \cdot 10^{-1}$	$2.551 \cdot 10^{-2}$
300	601	50	0.57	$8.142 \cdot 10^{-1}$	$1.605 \cdot 10^{-1}$	$2.531 \cdot 10^{-2}$
5	11	75	0.94	$6.048 \cdot 10^{-1}$	$2.740 \cdot 10^{-1}$	$1.212 \cdot 10^{-1}$
10	21	75	0.92	$5.993 \cdot 10^{-1}$	$2.861 \cdot 10^{-1}$	$1.146 \cdot 10^{-1}$
150	301	75	0.90	$5.939 \cdot 10^{-1}$	$2.984 \cdot 10^{-1}$	$1.077 \cdot 10^{-1}$
300	601	75	0.89	$5.937 \cdot 10^{-1}$	$2.989 \cdot 10^{-1}$	$1.074 \cdot 10^{-1}$
5	11	95	1.17	$4.691 \cdot 10^{-1}$	$3.267 \cdot 10^{-1}$	$2.043 \cdot 10^{-1}$
10	21	95	1.14	$4.592 \cdot 10^{-1}$	$3.420 \cdot 10^{-1}$	$1.988 \cdot 10^{-1}$
150	301	95	1.11	$4.491 \cdot 10^{-1}$	$3.579 \cdot 10^{-1}$	$1.930 \cdot 10^{-1}$
300	601	95	1.11	$4.487 \cdot 10^{-1}$	$3.585 \cdot 10^{-1}$	$1.928 \cdot 10^{-1}$
5	11	99	1.21	$4.457 \cdot 10^{-1}$	$3.334 \cdot 10^{-1}$	$2.209 \cdot 10^{-1}$
10	21	99	1.18	$4.353 \cdot 10^{-1}$	$3.490 \cdot 10^{-1}$	$2.157 \cdot 10^{-1}$
150	301	99	1.15	$4.247 \cdot 10^{-1}$	$3.651 \cdot 10^{-1}$	$2.102 \cdot 10^{-1}$
300	601	99	1.15	$4.243 \cdot 10^{-1}$	$3.657 \cdot 10^{-1}$	$2.100 \cdot 10^{-1}$
5	11	101	1.23	$4.345 \cdot 10^{-1}$	$3.363 \cdot 10^{-1}$	$2.291 \cdot 10^{-1}$
10	21	101	1.20	$4.239 \cdot 10^{-1}$	$3.520 \cdot 10^{-1}$	$2.241 \cdot 10^{-1}$
150	301	101	1.16	$4.131 \cdot 10^{-1}$	$3.682 \cdot 10^{-1}$	$2.188 \cdot 10^{-1}$
300	601	101	1.16	$4.127 \cdot 10^{-1}$	$3.688 \cdot 10^{-1}$	$2.186 \cdot 10^{-1}$
5	11	110	1.31	$3.879 \cdot 10^{-1}$	$3.466 \cdot 10^{-1}$	$2.655 \cdot 10^{-1}$
10	21	110	1.27	$3.766 \cdot 10^{-1}$	$3.622 \cdot 10^{-1}$	$2.611 \cdot 10^{-1}$
150	301	110	1.24	$3.651 \cdot 10^{-1}$	$3.784 \cdot 10^{-1}$	$2.565 \cdot 10^{-1}$
300	601	110	1.24	$3.647 \cdot 10^{-1}$	$3.790 \cdot 10^{-1}$	$2.563 \cdot 10^{-1}$
5	11	150	1.57	$2.419 \cdot 10^{-1}$	$3.527 \cdot 10^{-1}$	$4.055 \cdot 10^{-1}$
10	21	150	1.52	$2.311 \cdot 10^{-1}$	$3.657 \cdot 10^{-1}$	$4.033 \cdot 10^{-1}$
150	301	150	1.47	$2.202 \cdot 10^{-1}$	$3.788 \cdot 10^{-1}$	$4.010 \cdot 10^{-1}$
300	601	150	1.47	$2.198 \cdot 10^{-1}$	$3.793 \cdot 10^{-1}$	$4.009 \cdot 10^{-1}$
5	11	200	1.76	$1.464 \cdot 10^{-1}$	$3.236 \cdot 10^{-1}$	$5.300 \cdot 10^{-1}$
10	21	200	1.69	$1.382 \cdot 10^{-1}$	$3.327 \cdot 10^{-1}$	$5.291 \cdot 10^{-1}$
150	301	200	1.63	$1.301 \cdot 10^{-1}$	$3.419 \cdot 10^{-1}$	$5.281 \cdot 10^{-1}$
300	601	200	1.63	$1.298 \cdot 10^{-1}$	$3.422 \cdot 10^{-1}$	$5.280 \cdot 10^{-1}$

Table 16: Queueing model based capturing bulk service nature of block production

References

Heinrich Apfelmus, James Chapman, Carlos Tomé Cortiñas, Neil Davies, Javier Diaz, Yves Hauser, Andre Knispel, polinavinao, Ramsay Taylor, and Peter Thompson. Cardano formal specifications repository, August 2025. URL `https://github.com/IntersectMBO/cardano-formal-specifications`.

Bernardo David, Peter Gaži, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018*, pages 66–98, Cham, 2018. Springer International Publishing. ISBN 978-3-319-78375-8.

Seyed Hossein Haeri, Peter Thompson, Neil Davies, Peter Van Roy, Kevin Hammond, and James Chapman. Mind your outcomes: The delta-qsd paradigm for quality-centric systems development and its application to a blockchain case study. *Computers*, 11(3), 2022. ISSN 2073-431X. doi: 10.3390/computers11030045. URL `https://www.mdpi.com/2073-431X/11/3/45`.

Mikolaj Karpinski, Lyudmila Kovalchuk, Roman Kochan, Roman Oliynykov, Mariia Rodinko, and Lukasz Wieclaw. Blockchain technologies: Probability of double-spend attack on a proof-of-stake consensus. *Sensors*, 21:6408, 09 2021. doi: 10.3390/s21196408.

Lyudmila Kovalchuk, Roman Oliynykov, and Mariia Rodinko. Probability of double spend attack for pos consensus with ouroboros praos slot leader election procedure. In *CECC 2024 : 24th Central European Conference on Cryptology*.

Peter Van Roy, Neil Davies, Peter Thompson, and Seyed Hossein Haeri. The ΔQ SD systems development paradigm, a tutorial. In *HiPEAC Conference (High-performance Embedded Architecture and Compilation)*, Jun 2022. Tutorial.

List of Figures

List of Tables