Prediction of the Consolidation Delay in Blockchain-based Applications

Simonetta Balsamo
Andrea Marin
Università Ca' Foscari Venezia
Venice, Italy
{balsamo,marin}@unive.it

Isi Mitrani
University of Newcastle
Newcastle, United Kingdom
isi.mitrani@newcastle.ac.uk

Nicola Rebagliati
Blockchain Analyst
Lucca, Italy
nicola.rebagliati@gmail.com

ABSTRACT

In the last years, blockchains have become a popular technology to store immutable data validated in a peer-to-peer way. Software systems can take advantage of blockchains to publicly store data (organised in transactions) which is immutable by design. The most important consensus algorithm in public blockchains is the proof-of-work in which miners invest a huge computational power to consolidate new data in a ledger. Miners receive incentives for their work, i.e., a fee decided and paid for each transaction. Rational miners aim to maximise the profit generated by the mining activity, and thus choose the transactions offering the highest fee per byte for their consolidation.

In this paper, we propose a queueing model to study the relation between the fee offered by a transaction and its expected consolidation time, i.e., the time required to be added to the blockchain by the miners. The solution of the queueing model, although approximate, is computationally and numerically efficient and software systems can use it online to analyse the trade-off between costs and response times. Indeed, a static configuration of the model would not account for the high variations in the blockchain workload and fees offered by other users. The model takes into account the dropping of transactions caused by timeouts or finite capacity transaction pools. We validate our results with data extracted from the Bitcoin blockchain and with discrete event simulations.

CCS CONCEPTS

• **Computer systems organization → Peer-to-peer architectures**; • **Mathematics of computing → Markov processes**.

KEYWORDS

Blockchain, Queueing model, Consolidation time

ACM Reference Format:
Simonetta Balsamo, Andrea Marin, Isi Mitrani, and Nicola Rebagliati. 2021. Prediction of the Consolidation Delay in Blockchain-based Applications. In *Proceedings of the 2021 ACM/SPEC International Conference on Performance Engineering (ICPE '21), April 19–23, 2021, Virtual Event, France.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3427921.3450249

1 INTRODUCTION

Blockchain is a distributed ledger designed to store immutable data validated through a consensus protocol. In public, permissionless, blockchains the consensus protocol involves potentially all the ledger users in a pseudonymous way. Probably, the most well-known blockchain is that underlying the cryptocurrency Bitcoin (BTC). Most of the information stored in this system is related to pure transfers of cryptocurrency, but we should recall that we can store in the Bitcoin blockchain also other types of data like smart contracts or immutable signatures of documents.

The most common consensus protocol adopted in permissionless blockchains (but sometimes also in permissioned ones [16]) is called *Proof of Work* (PoW) and has been introduced in the seminal work of Nakamoto [4, 18]. While we leave to Section 3 a detailed description of the salient features of PoW, here we just informally introduce some important aspects.

The blockchain users send requests to the distributed ledger in which they ask to consolidate *transactions*. The consolidation of a transaction consists in its inclusion in one of the new blockchain blocks in an immutable way. Using the Bitcoin nomenclature, we call *MemPool* the set of transactions that have been sent to the blockchain but have not yet been consolidated. We will see that the Mempool behaves as a priority queue. *Miners* fetch the transactions from the local copy of the Mempool, validate them according to some rules (e.g., one cannot spend twice the same deposit), and try to solve a problem on the computation of the hashing. This problem is computationally expensive to solve but is very easy to verify. Once the problem has been solved, any other miner can verify the integrity of the information stored in the newly generated block and the process restarts.

Clearly, in order to incentivize the users to collaborate in mining, some reward mechanism must be planned to cover the energy and hardware costs that the miners have to face. This is a common problem for all systems with distributed validation (see, e.g., [9]). In Bitcoin, there are two types of rewards received by the miner who successfully consolidates a block: the first is an amount of cryptocurrency generated for this purpose and the second is the sum of the fees paid by the transactions. The first mechanism will progressively disappear as specified by the Bitcoin protocol. Therefore, the latter is gaining more and more importance. Since the block size is limited and the expected number of consolidated blocks per unit of time is approximately constant, the miners privilege the transactions offering the highest fees to maximise their profit.

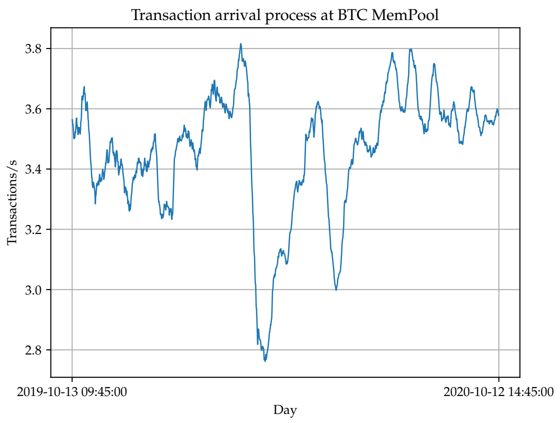


Figure 1: Intensity of the transaction arrival process from 2019 October, 13th to 2020 October, 12th. The plot shows the average obtained with a sliding window of 7 days. Data taken from `blockchain.com`.

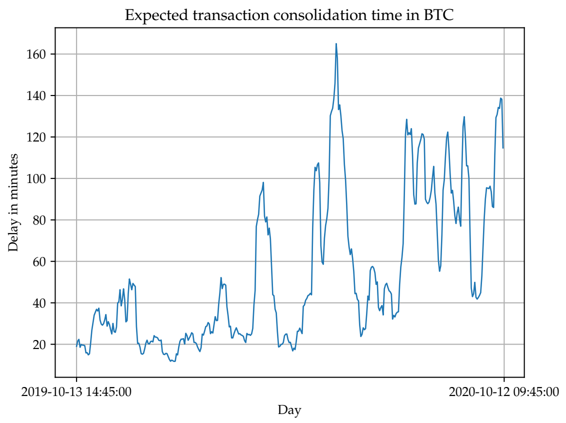


Figure 2: Expected transaction consolidation time from 2019 October, 13th to 2020 October, 12th. The plot shows the average obtained with a sliding window of 7 days. Data taken from `blockchain.com`.

As a consequence, the transition sojourn time in the MemPool depends on the offered fees: higher fees correspond to shorter consolidation times.

Software applications relying on PoW-based blockchains to store public immutable information must decide the fee that they intend to pay for their consolidation. Unfortunately, the connection between offered fees and expected consolidation time (also called confirmation time) depends on factors that are highly dynamic in the system, i.e., the distribution of the other users' offers, the intensity of the transaction arrival process at the MemPool and the availability of miners together with the level of difficulty of the PoW. The variability of the arrival process intensity in Figure 1 and Figure 2 shows that the average consolidation time for a transaction has varied from 30 to 160 minutes during the last year. Moreover, miners drop the transactions that stay in the MemPool more than a certain amount of time. This is needed to guarantee the stability of the mining process, i.e., from a queueing theoretical prospective, it guarantees the stability of the MemPool queue.

In this paper, we propose a queueing theoretical framework to support the decision policy at software level about the trade-off between cost reduction and system performance for what concerns the expected transaction consolidation time. The model that we propose is parameterised with the distribution of the fees offered by the blockchain users and the intensity of the arrival rate and allows the application to estimate the expected consolidation time associated with a certain fee. Figure 3 shows a sketch of an application running the proposed prediction system. A node will monitor and log the transactions arriving at the blockchain and communicates the intensity of the arrival process and the distribution of the fees to the model. The solution of the queueing model will provide estimates of the expected consolidation times for the application and will prepare its own transaction with the fee dynamically computed.

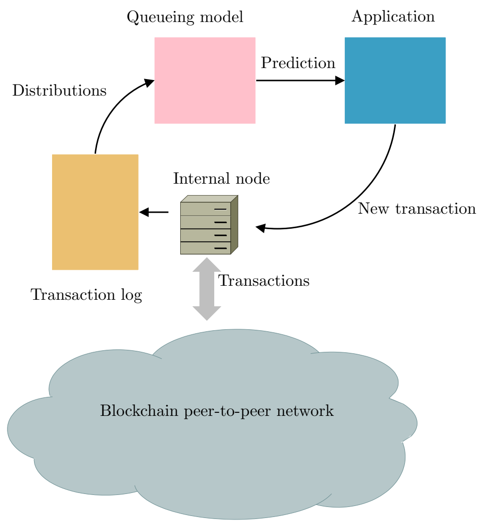


Figure 3: Structure of a software application with dynamic prediction of the expected transaction consolidation time.

We consider a queueing model with a set of jobs and with a priority scheduling. The analysis of sojourn time distribution for M/G/1 queues with priority is presented in [20], and an efficient computation algorithm of the steady-state probabilities, based on a matrix-geometrics method, is given for the M/M/1 queue with priority in [17]. Some works consider the analysis of the waiting time for the multiclass M/M/c queue with priority, with a closed form expression of the Laplace transform of the waiting time [8, 15], the average class sojourn time approximation in queues with preemption and different service rates [7], and an exact analysis, based on the generating function of the number of jobs with low priority, for the M/M/c with two priority classes [21].

Here, we consider a queuing model M/M^B/1 with bulk services of size B with hard priority and reneging for the class with the lowest priority. We provide an exact solution of the model without reneging and an approximate solution for the reneging case that has been validated by means of discrete event simulations.

The paper is structured as follows. Section 2 discusses some related work. Section 3 briefly describes the salient aspects of the mining process in PoW blockchains and emphasises the features that we model in this work. In Section 4, we describe the queueing model and its solution. In section 5, we use the BTC blockchain to validate our model. Specifically, we compare the predictions of our queueing model with the data available from the miners and stored in the chain. Section 6 shows some experiments with the reneging policy and, finally, Section 7 concludes the paper.

2 RELATED WORK

While the analysis of the blockchain system has drawn much attention from the research community for example with the development of simulators as [2], the analysis of the behaviour of a single transaction is still largely an open problem.

The users of a blockchain have an obvious need for a fee estimate and currently the Bitcoin network does not provide this service. There are independent services that try to give a rough estimate[1] and are based upon the MemPool state during the previous three hours and perform Montecarlo simulations. However, this method is quite slow for online predictions. Other services infer fees from the current MemPool state[2][3]. Finally, we note that third party services, e.g. cryptocurrency exchanges, ignore the fee problem by imposing very high fees to the users, independently of the network state. The importance of the impact of the miners' selfish decisions in a blockchain is studied in [1] by resorting to simulations.

In [12], the authors propose a simulator based on Omnet++ to study the relation between offered fee and consolidation time in Direct Acyclic Graph (DAG) blockchains. Although the purposes of the work are close to ours, they consider a different type of blockchain. In contrast with our contribution, [12] heavily relies on simulation while we develop an analytical model which is faster to solve and usable for online predictions.

In [13, 14], the authors propose a priority queueing model with batches with the same aims of ours. However, the two works differ for some important aspects. First, in [13, 14], when a miner includes a transaction in the block he is mining, he will not remove it, regardless to the fees of the following arrivals. In general, miners take advantage of the fact that the mining process is memoryless, i.e., the number of unsuccessful hashes computed on batch of transactions does not help the prediction the remaining work. Thus, whenever a high fee transaction arrives, it is better to change the candidate block composition. This has important consequences for the heavy-load case. Since high fee transactions arrive according to an independent process at the miner MemPool, they will find the block being mined full with high probability. Thus, if not allowed to replace existing low fee transactions, they will have to wait until the next block for consolidation. The experiments in Section 5, show that, even under heavy load, the expected number of blocks that a high fee transaction have to wait is slightly above 1 (we discuss in the same section why it is not exactly 1). In our model, the miners always choose the transactions with the highest fee when they mine a block. A second difference is that we consider the possibility that low priority queues receive insufficient service capacity, and examine mechanisms for avoiding their saturation. Thirdly, we compare the prediction of our models both with discrete event simulation and with real data obtained by the BTC blockchain.

[1]https://bitcoinfees.earn.com
[2]https://twitter.com/bitcoin_fees
[3]https://btc.network/estimate

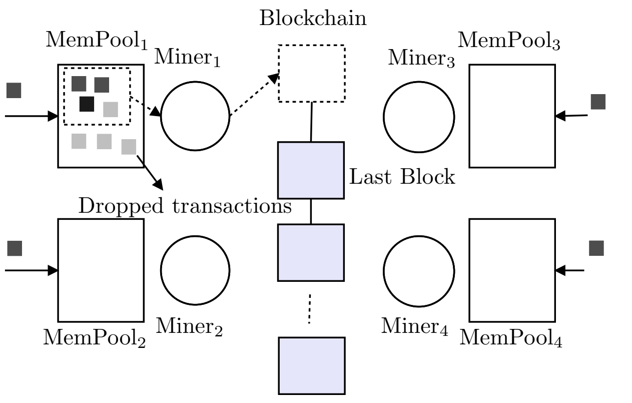


Figure 4: Sketch of the mining process. The color of the boxes representing the transactions in the MemPool denotes their fees: the darkest colors correspond to the highest fees.

3 BACKGROUND ON POW IN BLOCKCHAIN NETWORKS

Blockchains store immutable transactions aggregated in blocks. Transactions are *confirmed* thanks to a consensus protocol that characterises the blockchain. In this work, we focus on the widely used PoW consensus protocol as described in the seminal Nakamoto's paper [18]. Although this has been developed for one specific blockchain, i.e., that underlying the cryptocurrency Bitcoin, the same idea can be found in many others public blockchains.

From the quantitative point of view, these blockchains impose a maximum block size and an average delay between blocks. These invariant properties are required for security reasons and determine the maximum throughput of the system, defined as the maximum number of transactions that can be added to a block per unit of time. For example, in BTC, the maximum block size is 1MB (which approximately correspond to $2,800$ transactions) and the blocks are generated, on average, every 10 minutes.

3.1 The mining process

We can summarise the PoW consensus protocol as follows. Each miner, which can be any user, is connected to the others to form a peer-to-peer network. It maintains a queue with the pending transactions, i.e., the transactions that have to be added to a block. This queue is called MemPool in BTC, and we will follow this nomenclature for simplicity. The miner selects the transactions to include in the block and computes the hash of the set of transactions plus several other information among which the *nonce* plays an important role. Indeed, the problem of the PoW is that of finding a nonce that produces a block hash with certain characteristics, e.g., that begins with a string of 0s as in BTC. The network self-adjusts the requirements on the block hashes so that the total hash-power of the network can produce on average one block every ΔT seconds, where $\Delta T = 600$s for BTC.

Figure 4 shows a sketch of this process. Notice that, although the contents of the miners' MemPools may be different, these differences are very small and in the model of Section 4 we will consider them to be identical.

Due to the memoryless nature of the hash computation, the miners may decide to change the candidate block transactions in any moment during the mining without affecting the probability of successfully finding the correct nonce. Moreover, the same property allows us to conclude that the time between two block consolidations by a miner must be approximately exponentially distributed and, by the independence of the miners' mining processes, the delay between the consolidations of two blocks in the network is also exponentially distributed with rate $1/600s^{-1}$, i.e., with an average distance of 10 minutes. See, e.g., [10] for an experimental validation of this observation.

The way miners choose the transactions to put in the block determines the scheduling discipline of the MemPool. At this point, we have to say that BTC and many other ledgers do not specify any rule about which transactions have to be added to the pool and when they have to be dropped due to timeout. As long as a block contains valid transactions, it will be accepted by the other miners. While we omit the explanations of the validation process and the solution of possible forks in the network, it is important to understand the policy adopted by the miners to choose the transactions from the MemPool that are going to be consolidated.

Since miners must face a cost for their work (hardware and energy consumed during the mining process), they are rewarded whenever they consolidate a block. There are several ways of doing this in blockchains, but the most common is allowing transactions to offer a fee for their consolidation. When the blockchain is associated with a cryptocurrency, this fee is paid with that cryptocurrency. Given the limited amount of space in the blocks, miners tend to maximise their profit by including in the block the transactions that offer the highest fee per byte. This behaviour induces a priority of transactions with higher fees over the other ones. Moreover, it is worth of noticing that by the memoryless property of the hash computation, whenever a transaction with high fee arrives at a miner, the most convenient choice for him is to replace the transaction with the lowest fee in his candidate block with the newly arrived one. We stress on the fact that, in general, this is not required by the protocols but it is safe to assume that miners will try to maximise their profit.

Finally, we discuss the dropping policy applied by the miners. Recall that each miner has a private copy of the MemPool and he is only required to include valid transactions in the block, no matter the order of arrival. A similar rule applies to the dropping policy. When the intensity of the arrival process is higher than the maximum ledger's throughput, the MemPool size grows because the queueing process becomes unstable. In order to avoid this problem, a dropping policy is adopted by the miners. The strategy may be either to drop the oldest transactions when the MemPool reaches a certain size, or to drop the transactions whose residence time in the MemPool exceeds a certain timeout. In some cases, both the approaches are used together. However, usually the protocols do not impose the dropping timeout or the MemPool size for the miners who are allowed to choose their own configurations. Nevertheless, there are some recommended values for these parameters which are used as a default configuration for mining software and hence used by most of the miners.

4 QUEUEING THEORETICAL ANALYSIS

In this section, we present the queueing model for the mining process and its solution. The accuracy of these results will be studied in Section 5.

4.1 The queueing model

We consider a system with K job types, numbered $1, 2, \ldots, K$. Jobs of type i arrive according to an independent Poisson process with rate λ_i, and join a separate First-In-First-Out (FIFO) queue. Another independent Poisson process, with rate μ, defines a sequence of 'consolidation' instants, at which up to B jobs at a time are removed from the system. It is assumed that jobs of type i pay higher fees for service than those of type j if $i < j$. Consequently, when filling a consolidation batch, queue i is given priority over queue j. Thus, if there are B or more jobs in queue 1, then the entire batch is filled with type 1 jobs; otherwise, the remaining spaces are filled with type 2 jobs, if available; etc. If, at a consolidation instant, the total number of jobs in the system does not exceed B, then all queues are emptied.

The above assumptions imply that the stability condition for queue i is that the total arrival rate into queues $1, 2, \ldots, i$, $\Lambda_i = \lambda_1 + \lambda_2 + \ldots + \lambda_i$, is lower than the maximum consolidation rate:

$$\Lambda_i < B\mu \, . \tag{1}$$

We are interested in the case where some queues are stable, but possibly not all. That is, there is an index m, $1 \leq m \leq K$, such that queues $1, 2, \ldots, m$ are stable, while queues $m+1, m+2, \ldots, K$, if any, would be unstable and would grow without bound unless some job filtering mechanism is employed. Two such mechanisms deserve consideration.

(a) Random reneging: An incoming job of type $i > m$ starts an interval timer distributed exponentially with mean $1/\gamma_i$; if that timer expires before the job is consolidated, it leaves the queue and is lost.
(b) Baulking: A bound N is imposed on the size of queue $i > m$; any type i job that finds N jobs already present in that queue, refuses to join and is lost.

The object of the analysis is to determine the average response times, $W_1, W_2, \ldots, W_K$, for jobs of different types. When $i > m$, the corresponding average is conditioned upon the job joining its queue, or not reneging. Those performance measures may help users to decide whether to pay higher fees in order to improve their response time.

4.2 Exact solution for queues $1, 2, \ldots, m$

Consider first queue 1. This is known in the literature as an M/M^B/1 queue with arrival rate λ_1 and bulk services of size B, occurring at rate μ. It was first analysed by Bailey [5] in the context of general inter-service intervals, and has since been examined in a variety of other settings. For completeness, we shall present here a new and simple derivation of the result, while the study of the priority and reneging polices is novel.

Let $p_{1,n}$ be the steady-state probability that there are n jobs in queue 1. Equating the upward and downward flows across queue level n, the corresponding balance equations can be written as follows.

$$\lambda_1 p_{1,n} = \mu \sum_{j=n+1}^{n+B} p_{1,j} \quad ; \quad n = 0, 1, \dots \; . \tag{2}$$

These equations have a geometric solution of the form

$$p_{1,n} = C z_1^n \; , \tag{3}$$

where C and z_1 are some positive constants. Indeed, substituting (3) into (2), we find that the equations are satisfied as long as z_1 is a zero of the polynomial of degree B

$$P_1(z) = \lambda_1 - \mu \sum_{j=1}^{B} z^j \; . \tag{4}$$

In other words, z_1 must be a zero of the polynomial

$$Q_1(z) = \lambda_1(1 - z) - \mu z(1 - z^B) \; . \tag{5}$$

In addition, in order that we may obtain a probability distribution, z_1 must be in the interior of the unit disk, $|z_1| < 1$.

Note that $Q_1(0) > 0$ and $Q_1(1) = 0$. However, $Q_1'(1) = B\mu - \lambda_1 > 0$. Therefore, $Q_1(1 - \epsilon) < 0$ for some sufficiently small ϵ. Hence, $Q_1(z)$ has a real zero, z_1, in the interval $(0, 1 - \epsilon)$. Moreover, it can be shown that $Q_1(z)$ has no other zeros in the interior of the unit disk. That follows from Rouche's theorem.

Thus, the steady-state distribution of the number of jobs in queue 1 is given by

$$p_{1,n} = (1 - z_1) z_1^n \quad ; \quad n = 0, 1, \dots \; . \tag{6}$$

Similarly to the M/M/1 queue, the average number of jobs in queue 1, L_1, is given by

$$L_1 = \frac{z_1}{1 - z_1} \; . \tag{7}$$

The average response time of a type 1 job, W_1, is obtained from Little's theorem: $W_1 = L_1/\lambda_1$.

Now consider queues 1 and 2. We make the following observation: if, instead of giving priority to type 1 jobs when filling the service batch, some other policy, e.g. FIFO was employed, the total average number of jobs in the two queues would not change (provided, of course, that those two types have priority over all others). That total average number of jobs, which we shall denote by $L^{(2)}$, can therefore be obtained by lumping together type 1 and type 2 jobs into one bulk service queue with arrival rate $\Lambda_2 = \lambda_1 + \lambda_2$, and applying the above solution. This yields

$$L^{(2)} = \frac{z_2}{1 - z_2} \; , \tag{8}$$

where z_2 is the single zero in the interval (0,1) of the polynomial

$$Q_2(z) = \Lambda_2(1 - z) - \mu z(1 - z^B) \; . \tag{9}$$

Returning to the original priority policy, and noting that the average number L_1 of type 1 jobs has already been determined, we can now find the average number L_2 of type 2 jobs present.

$$L_2 = L^{(2)} - L_1 \; . \tag{10}$$

The average response time for type 2 jobs is $W_2 = L_2/\lambda_2$.

In general, if the total average number of jobs in queues 1, 2, ..., i, $L^{(i)}$, has been computed, the average size of queue $i + 1$ is obtained by first evaluating the total number of jobs in the $i + 1$ queues, $L^{(i+1)}$, using the zero z_{i+1} of the polynomial (9), with Λ_2 replaced by Λ_{i+1}. Then

$$L_{i+1} = L^{(i+1)} - L^{(i)}. \tag{11}$$

The average response time in queue $i + 1$ is $W_{i+1} = L_{i+1}/\lambda_{i+1}$. This process is valid as long as the corresponding queues are stable.

4.3 Low priority queues

We have assumed that queue m is stable, but queue $m + 1$ is not. In other words, there is some service capacity left over after serving the higher priority queues, but it is insufficient to cope with the offered load of type $m + 1$. Hence, some lower priority jobs must be dropped in order to prevent their numbers from growing without bound. One possibility would be to devote all spare capacity to queue $m + 1$, control its size by one of the two filtering mechanisms suggested earlier, and drop all jobs of types $m + 2$, $m + 3$, ..., K. Alternatively, queues $m + 1$, $m + 2$, ..., K could be lumped into a single queue, ℓ, and apply one of the filtering mechanisms.

From the modelling point of view, the only difference between the above two alternatives is that in the first case the arrival rate into the lower priority queue is λ_{m+1}, while in the second case it is $\Lambda_\ell = \lambda_{m+1} + \lambda_{m+2} + \lambda_K$. We choose to analyse the second alternative because it offers jobs of all lower priority types the possibility of being consolidated.

To get a handle on the performance that the lower priority jobs may expect, we have to make some approximations. The idea is to model the jobs of type ℓ as being served one at a time, but choosing their average service time appropriately, to reflect the service capacity available to them.

Note that the instants at which batches of waiting jobs are removed from the system, form a Poisson process. Therefore, by the PASTA property, a batch about to be filled sees the steady-state distribution of the system state. In particular, it sees the steady-state distribution of the total number of higher priority jobs. The probability, p_n, that there is a total of n jobs of types 1, 2, ..., m present, is given by

$$p_n = (1 - z_m) z_m^n \quad ; \quad n = 0, 1, \dots \; , \tag{12}$$

where z_m is the single zero in the interval (0,1) of the polynomial

$$Q_m(z) = \Lambda_m(1 - z) - \mu z(1 - z^B) \; . \tag{13}$$

This allows us to determine the average number, b, of spaces available in the batch for lower priority jobs.

$$b = \sum_{n=0}^{B-1} (B - n) p_n \; . \tag{14}$$

Hence, the average service rate available to jobs of type ℓ is μb. Our approximation consists in assuming that those jobs are served one at a time, and their service times are distributed exponentially with mean $1/\nu = 1/(\mu b)$. The performance they observe depends on the filtering mechanism employed.

4.3.1 Reneging. Queues with reneging have been studied quite extensively. A survey of related literature can be found in Bocquet [6]. In particular, the case of a single server with random reneging appears to have been solved first by Ancker and Gafarian [3]. However, the results that we have seen are not suitable for models where the arrival and service parameters are large (see comment below). We provide expressions that lead to numerically stable computations.

Jobs of type ℓ arrive at rate Λ_ℓ and queue in FIFO order. Services are i.i.d. random variables distributed exponentially with mean $1/\nu$. While waiting, each job reneges at rate γ. Thus, queue ℓ evolves as a Birth-and-Death process with birth rate Λ_ℓ and death rate, when there are n jobs present, $\nu + n\gamma$. The steady-state probabilities, π_n, satisfy the balance equations

$$\Lambda_\ell \pi_n = (\nu + (n+1)\gamma)\pi_{n+1} \quad ; \quad n = 0, 1, \dots \; . \tag{15}$$

Let $g(z)$ be the generating function

$$g(z) = \sum_{n=0}^{\infty} \pi_n z^n \; . \tag{16}$$

Multiplying equation (15) by z^n and summing, we obtain

$$\Lambda_\ell g(z) = \frac{\nu}{z}[g(z) - \pi_0] + \gamma g'(z) \; , \tag{17}$$

which can be rewritten as

$$g'(z) + [\frac{\eta}{z} - \sigma]g(z) = \frac{\eta}{z}\pi_0 \; , \tag{18}$$

where $\eta = \nu/\gamma$ and $\sigma = \Lambda_\ell/\gamma$. This is an ordinary linear differential equation of first order. Its general solution has the form

$$g(z) = z^{-\eta} e^{\sigma z} \left[\pi_0 \eta \int_0^z y^{\eta-1} e^{-\sigma y} dy + C \right] \; , \tag{19}$$

where C is an arbitrary constant. In our case, we must have $C = 0$, in order that $g(0) = \pi_0$.

The unknown probability π_0 is determined from the normalizing equation $g(1) = 1$. This yields

$$\pi_0 = e^{-\sigma} \left[\eta \int_0^1 y^{\eta-1} e^{-\sigma y} dy \right]^{-1} \; . \tag{20}$$

The integral in the right-hand side is related to the lower incomplete gamma function. When σ and η are large (e.g., in the thousands), a straightforward evaluation of that integral can produce numerical indeterminacies of the type $0 \cdot \infty$. A more stable computation is achieved by rewriting Eq. (20) in a form that replaces multiplications by additions:

$$\pi_0 = \left[\eta \int_0^1 e^{(\eta-1)\log(y) + \sigma(1-y)} dy \right]^{-1} \; . \tag{21}$$

The average number of type ℓ jobs in the queue, L_ℓ, is given by $g'(1)$ which, according to (18), is equal to

$$L_\ell = \sigma - \eta(1 - \pi_0) \; . \tag{22}$$

Now we wish to estimate the probability, $q(L)$, that a job of type ℓ is consolidated before it reneges, given that there are L jobs of type ℓ ahead of it in the queue. In order to be consolidated, the tagged job must survive (i.e., not renege) until the next service epoch. Then, if not consolidated, it must survive until the following service epoch, and so on, until eventually it is consolidated.

Since both the service and the reneging intervals are distributed exponentially, with parameters μ and γ respectively, the probability that the tagged job survives until the next service epoch is $\mu/(\mu+\gamma)$. Given that it does survive, the average interval until that epoch is distributed exponentially with mean $1/(\mu + \gamma)$.

While the tagged job is waiting, the number of type ℓ jobs ahead of it decreases, due to renegings. We shall treat that number as a deterministic fluid whose rate of decrease is proportional to the amount present. That is, if there are no services, at time t after the arrival of the tagged job, the number, $L(t)$ of type ℓ jobs ahead of it is given by

$$L(t) = Le^{-\gamma t} \; . \tag{23}$$

Consequently, at the next service epoch, the average number of type ℓ jobs ahead of the tagged one would be

$$\tilde{L} = \int_0^\infty L(t)(\mu + \gamma)e^{-(\mu+\gamma)t} dt = \frac{L(\mu + \gamma)}{\mu + 2\gamma} \; . \tag{24}$$

Remember that in a service batch there are, on the average, b spaces available to type ℓ jobs as given by Eq. (14). Consequently, we shall assume that if $\tilde{L} < b$, the tagged job only needs to survive until the next service epoch in order to be consolidated:

$$q(L) = \frac{\mu}{\mu + \gamma} \quad ; \quad \tilde{L} < b \; . \tag{25}$$

On the other hand, if $\tilde{L} \geq b$, after the next batch the tagged job will face a situation where $\tilde{L}$ is reduced by b. Hence, we may write

$$q(L) = \frac{\mu}{\mu + \gamma} q(\tilde{L} - b) \quad ; \quad \tilde{L} \geq b \; . \tag{26}$$

Expressions (24) – (26) allow us to compute the desired $q(L_\ell)$ recursively. The average response time of successful jobs of type ℓ is estimated by using Little's theorem:

$$W_\ell = \frac{L_\ell}{\Lambda_\ell q(L_\ell)} \; . \tag{27}$$

4.3.2 Bounded queue. Jobs of type ℓ arrive at rate Λ_ℓ and join the queue if there are fewer than N jobs present. Services are i.i.d. random variables distributed exponentially with mean $1/\nu$. In other words, queue ℓ behaves like an M/M/1/N queue. Its steady-state distribution is given by

$$\pi_n = \frac{(1 - \rho_\ell)\rho_\ell^n}{1 - \rho_\ell^{N+1}} \quad ; \quad n = 0, 1, \dots, N \; , \tag{28}$$

where $\rho_\ell = \Lambda_\ell/\nu$ is the offered load. That parameter would, in our case, be larger than 1. If $\rho_\ell = 1$, then (28) becomes $\pi_n = 1/N$.

The probability, q, that an incoming job joins the queue, and is therefore consolidated, is given by

$$q = 1 - \frac{(1 - \rho_\ell)\rho_\ell^N}{1 - \rho_\ell^{N+1}} \; . \tag{29}$$

The average number of type ℓ jobs present, L_ℓ, is computed from

$$L_\ell = \sum_{n=1}^{N} n\pi_n \; . \tag{30}$$

Finally, the average response time, W_ℓ, of a type ℓ job that joins the queue, can be approximated by Little's theorem as:

$$W_\ell = \frac{L_\ell}{\Lambda_\ell q} . \tag{31}$$

It should be pointed out that when $\rho_\ell > 1$ and N is large, say in the hundreds of thousands, expressions (28) – (30) are difficult or impossible to evaluate numerically. In such cases, a stable computation is achieved by setting $\pi_N = 1$, evaluating $\pi_i = \pi_{i+1}/\rho_\ell$ for all i, and normalizing.

5 VALIDATION WITH BTC BLOCKCHAIN

The queueing model of Section 4 is an abstraction of the mining process. Specifically, there are some aspects that have been abstracted out as well as some approximations that have been introduced to allow an efficient numerical tractability of its solution.

In this section, we study the accuracy of the model predictions with respect to the data available in BTC blockchain.

5.1 Methodology

We collect all the transactions consolidated in blocks belonging to a certain time interval. The time interval contains approximately 40 blocks: longer intervals have the problem of the inhomogeneity of the arrival process whose intensity varies along the day while shorter intervals contain too few data to have a robust estimate of the consolidation time. For each transaction, we search the *first seen* timestamp obtained by the nodes of `blockchain.com` and `bitaps.com` and compute the expected consolidation time for each class. The intensity of the arrival process is measured or obtained from the historical data of the chain. To reduce the variance of the consolidation times, we measure the number of blocks required for the consolidation of a transaction, rather than the time delay. This is done to remove the variance due to the exponential distribution describing the inter-arrival times of new blocks, since we cannot measure much more than 40 blocks while maintaining the same conditions of the arrival intensity and fee distribution. All the times that we show in the following sections are expressed in UTC. In Bitcoin, fees are usually expressed in terms of Satoshi per Byte (S/B), where 1 BTC corresponds to 10^8 Satoshi.

5.2 Sources of noise

The implementation of the mining process described in Section 3 introduces some noise in our measurements. For example, we expect that transaction with very high fees will always be consolidated within one block, but we will see this is not true and the average is slightly higher than 1. In what follows, we describe some behaviours of the BTC network that explain why the measurements are not exactly what we could expect.

- Transaction may arrive at the system immediately before the block mining, thus the miners are unable to update the transaction list in the candidate block in time for its inclusion. For example, transaction t^4 is seen by `bitaps` node at 2020-10-15 16:39:09 UTC and the successive block (height 652885) is mined at 2020-10-15 16:39:10 but the transaction is not included. In our measurements, we observed that this happens when the block is mined within 30s from the arrival of a high fee transaction. Moreover, recall that the peer-to-peer network is designed in such a way that there is a propagation delay of transactions. The pdf of the propagation delay is shown in Figure 5 and we may see that most of the transactions propagate in less than 5 seconds.
- Another source of noise is connected with different policies applied by the miners. For example transaction t^5 is consolidated after 12 blocks although is offering a very high fee that should lead to the consolidation in 1 or at most 2 blocks. A possible explanation is connected to the fact that the transaction is marked with *version 2* while the current official version for transactions is 1. Thus, some miners may decide not to add it to their candidate blocks hence delaying its consolidation. However, these cases are very few: we found 3 cases of transactions offering more than 400 S/B and marked with version 2 with delay higher than 2 blocks.
- We should consider the effects of the greedy miner attack on the measurements on the chain. This kind of attack to the blockchain does not affect the integrity of its content but the way the fees are distributed among the peers. It works as follows: when the greedy miner (typically a mining pool) mines a block, it does not announce it to the network, and keeps mining the successive block. When one of the other honest miners announces a new block, the greedy miner announces its own. The blockchain has a fork which is solved when the next block will be appended to one of the two chains. The implications of the greedy miner attack is that, if this is successful, the node added by the greedy miner does not contain the high fee transactions arriving during the time interval between the mining of that block and the mining of the honest miners' block [11]. Moreover, internal selfish behaviours in mining pools may have the same effects [22]. These behaviours cause more forks to the blockchain than those expected by the contemporaneity of the mining announcements by two miners and as a consequence low fee transactions are favoured more than expected.
- Sometimes, transactions with high fee per byte depend on transactions with low fees per byte that have not been consolidated yet. Thus, miners are forced to include the low fee transactions to gain the revenue of the high fee one.

5.3 Validation in heavy-load

In the first test, we monitored the system from October/15/2020 16:03:12 to October/15/2020 18:31:20 and measured an arrival intensity of 4.3 jobs per second. We chose to divide the fees in 5 classes according to Table 1. The maximum block size is set to $2,800$ transactions. The load factor of the system in this experiment was 92%.

5.4 Validation in moderate-load

In the second test, we monitored the system from October/12/2020 07:44:22 to October/12/2020 12:12:49 and measured an arrival intensity of 3.5 jobs per second. The classes are the same of Table 1 but

[4]txid: ac64fb1d7b1ee7d82a1a6b3c9ae0fe8c088fb299b68066b82ce2fa5257a5a4f3

[5]txid: e4a3b02bff2479c1b18f4e97e52a224ddb00121e2006d7afb973936955c2f9a8

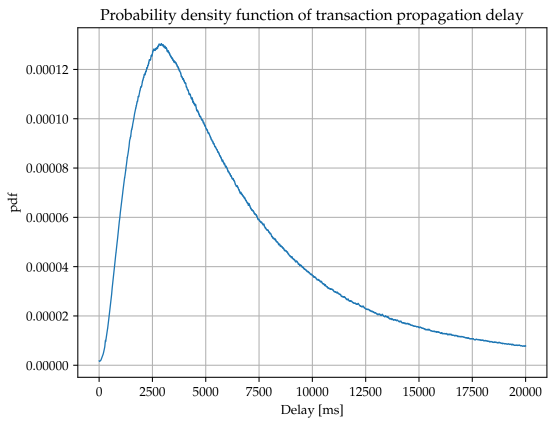


Figure 5: Probability density function (pdf) of the transaction propagation delay. Source: [19] for 2020-10-16.

Class	Range [S/B]	Dist. in high-load	Dist. in moderate-load
1	$[100, \infty)$	0.069	0.066
2	$[60, 100)$	0.235	0.216
3	$[40, 60)$	0.315	0.152
4	$[20, 40)$	0.184	0.096
5	$[0, 20)$	0.196	0.470

Table 1: Classes and frequencies for the heavy-load and moderate-load test.

clearly the frequencies are different since users tend to offer higher consolidation fees when the load factor is high. The maximum block size is the same as before, $2,800$ transactions. The load factor of the system in this experiment was 75%.

5.5 Discussion

The model and the simulation estimate the expected consolidation time in an accurate way as shown by Figures 6a and 6b. We note that both the model and the simulation underestimate the expected consolidation time for high fee transactions. We believe this is a consequence of the observations explained in Section 5.2; the model and the simulation catch very well the fact that very low-fee transactions tend to be delayed much more than the others. In other words, especially in moderate-load, there is not much difference in belonging to class 1, 2 or 3, 4 while class 5 shows a huge increase of the expected consolidation time. In heavy-load, class 4 offers an expected consolidation time which is approximately three times that of the best class while lass 5 is highly slowed down.

In Figure 7, we show the consolidation time distribution obtained with the stochastic simulation. It is instructive to see that higher priority classes consolidation time have approximately the same distribution in high and moderate-load, while lower priority ones tend to be heavily tailed especially in high-load. These considerations are useful if the application has service level agreements on deadlines rather than on averages. Our analytical model does not provide the distributions, but nevertheless the application should be aware that with lower priority classes the distribution of the consolidation time becomes heavy-tailed as well as with a higher expectation.

6 EXPERIMENTS

In this section, we study the model under extreme scenarios.

In Section 6.1 we study the impact of the intensity of the arrival rate on the expected consolidation times of the classes. However, we should stress on the fact that the intensity of the arrival process is correlated with the fees offered by the end users as discussed in Section 5. However, the importance of this experiment relies on the observations that we can draw on the behaviour of the consolidation time for low priority classes.

In Section 6.2, we study blockchains that adopt reneging policies based on timeouts. In this case, since the queueing model described in Section 4 is solved by resorting to an approximate method, we compare the predictions with the estimates obtained with simulation.

Section 6.3 studies the system with transaction droppings caused by the saturation of the MemPool.

Finally, Section 6.4 gives some details about the simulations performed.

6.1 Impact of the arrival rate

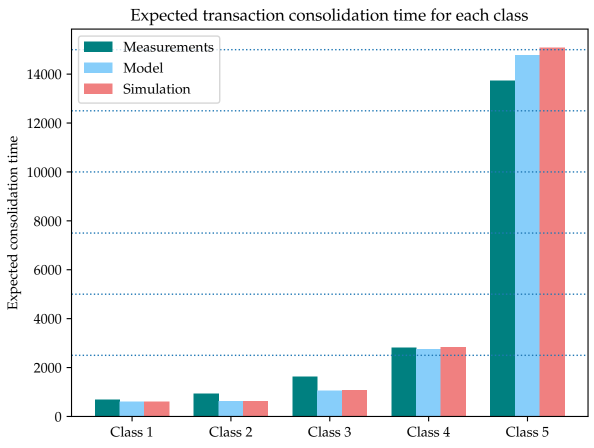
In this experiment, we consider the distribution of the jobs seen for the moderate-load validation (see Table 1) and vary the intensity of the arrival process from 1 to 4.4 transactions/s. Notice that the queue is stable if the total arrival rate is lower than

$$\frac{2800}{600} \simeq 4.66 \text{ transactions/s},$$

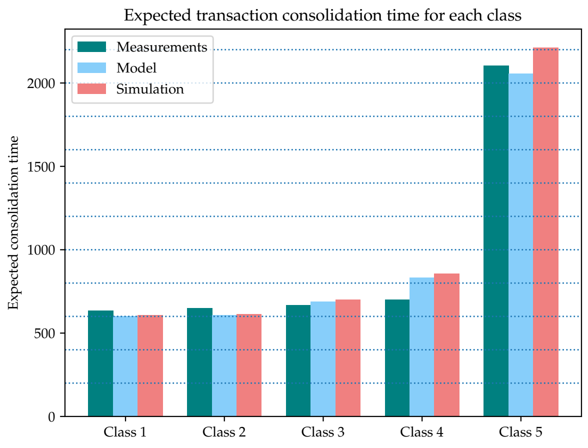
Therefore, we are very close to the saturation of the queue. The frequencies of the classes do not change. Since for this case the model is exact, we do not compare the results with simulations. Figure 8a shows the expected consolidation time for the 4 classes with highest priority, while Figure 8b shows the expected consolidation time for the lowest priority class. Notice that the vertical axes of the two plots have different scales. We may see that, as expected, the increase of the workload has the most evident effects on the lowest priority class. The first two classes are almost unaffected by the heavy-load regime. This clearly explains why, in practice, transaction fees increase during heavy loads, and the importance of predicting the consolidation time in an automatic way. Figure 8 clearly shows the vertical asymptote associated with the saturation of open queues.

6.2 Impact of reneging on the lowest priority class with timeout

In this experiment, we consider the five classes of the moderate-load case shown in Table 1 and imagine that there is a sixth class with even lower priority. The arrival rate of each class from 1 to 5 is given by the product of the moderate-load intensity (3.5 transactions per second) and the frequency associated with that class. Class 6 arrival rate ranges from 1.25 to 3 transaction per seconds. Notice that the new class would make a queue without timeout grow indefinitely since the arrival rate becomes higher than 4.66 transaction/s. Thus,



(a) Validation in heavy-load: expected transaction consolidation time in seconds.



(b) Validation in moderate-load: expected transaction consolidation time in seconds.

Figure 6: Validation of model and simulation with BTC blockchain

the reneging mechanism is required to maintain the MemPool with a finite population with probability 1.

The timeout is exponentially distributed with mean 72 and 48[6] hours for the first (Figures 9a and 9b) and second (Figures 9c and 9d) test, respectively.

Figures 9a and 9c show the expected occupancy of the MemPool limited to the jobs of class 6. Clearly, the analysis done in Section 5 is still valid here, and hence we focus only on the class with the lowest priority. For what concerns the accuracy of the approximated model analysis, we can see that it is in general very good with the exception of the case in which the intensity of the arrivals is slightly above the maximum system service capacity. Indeed, this is the case in which the approximation of Section 4 is less accurate because of the replacement of the batch service with single one.

We note that the increase of the occupancy (and hence of W) is apparently linear. This is not really surprising, although we had not predicted it before performing the experiment. What seems to be happening is that the usual non-linear increase in the queue size is compensated by a non-linear increase in reneging. In retrospect, perhaps we could have come to that conclusion, but even if we had decided to postulate a linear relationship, an analysis would have been necessary in order to determine its parameters.

A final, important, observation is that while the dropping probability is very similar for 72 and 48 hours of expected timeout, the expected occupancy of the MemPool is lower in the latter case. Thus, our experiments suggest that blockchains that use a timeout-based dropping mechanism should carefully evaluate the trade-off between miners' resource usage and dropping probability. In this scenario, 48 hours appear to be a better choice than 72 hours.

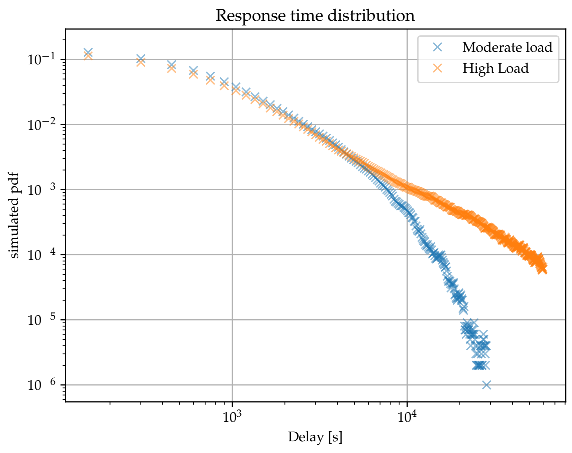
[6]See `https://bitcoin.org/en/bitcoin-core/`

6.3 Impact of reneging on the lowest priority class with bounded queue

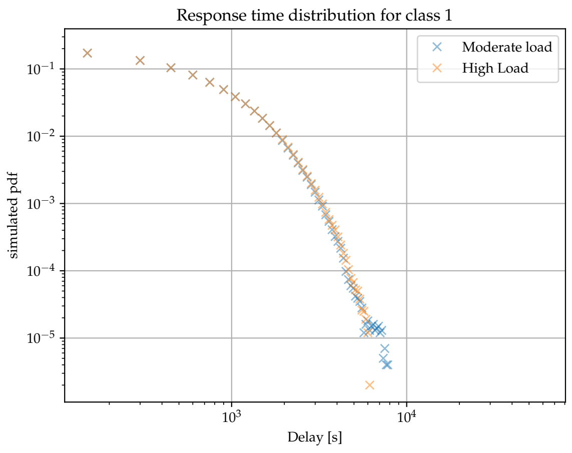
The last experiment considers the system with reneging based on the finite capacity of the MemPool. We fix a capacity of $5 \cdot 10^5$ transactions for the lowest priority class. Arrival rate and class frequencies are the same of those presented in Section 6.2. Figure 9e and 9f show the expected buffer occupancy and the consolidation probability for class 6 transactions, respectively. For what concerns the accuracy of the model, we can see that the highest discrepancy can be observed for the lowest arrival rate. Also in this case, this happens because we approximate the batch service discipline with a single job one to maintain the analytical tractability. The figures show that, for higher loads, the buffer tends to be always full while the consolidation probability is similar to that observed in the experiments of Section 6.2. Thus, the finite buffer solution offers approximately the same performance in terms of probability of consolidation of the other solutions but its expected resource demand is higher. The advantage is that the miner has pre-determined and well-known resource demand, while with the timeout policy, he has to face the variance of the MemPool occupancy.

6.4 Note on the simulation methodology

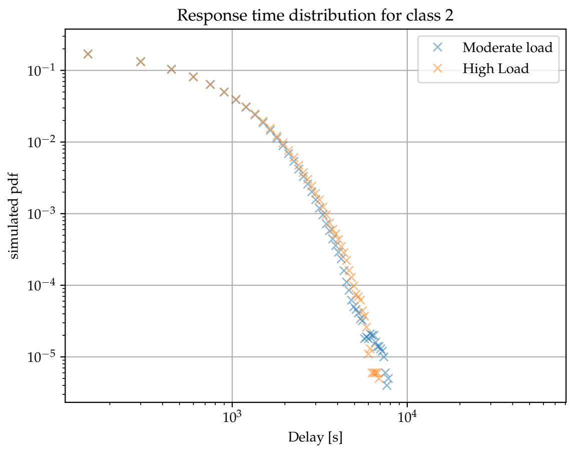
When necessary, we have resorted to discrete event simulators to validate our analytical results. Since, the system we are studying is heavily loaded in most of the cases, simulations require long time to converge and may exhibit numerical problems. For these reasons the simulations have been cross validated by two simulators, where the first simulates the Markov process underlying the system and the second the detailed customer behaviours. It is worth of notice that to reach the stationary regimes and to have confidence intervals of 95% within 5% of relative error in the estimates, we had to simulate the arrival of over $2 \cdot 10^8$ transactions.



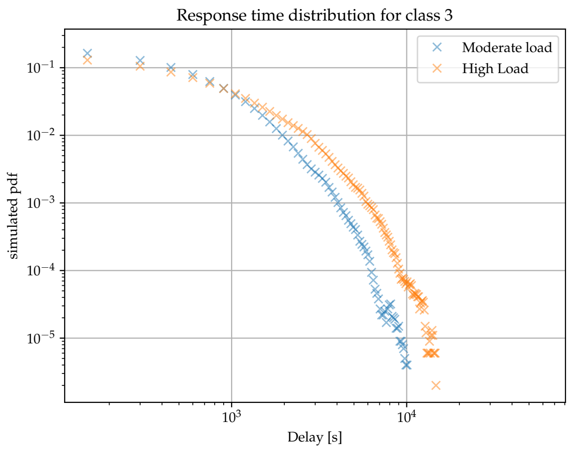
(a) Overall simulated response time distribution .



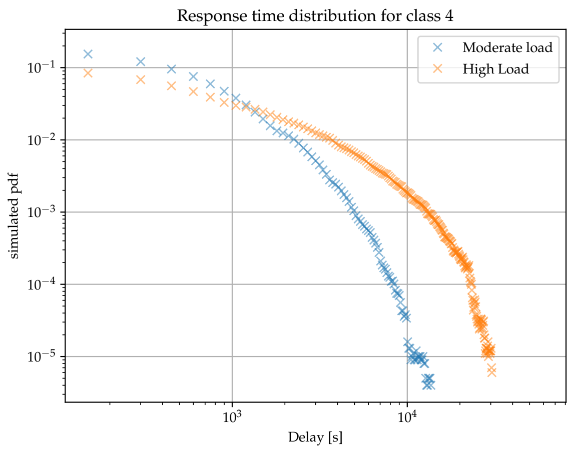
(b) Simulated response time distribution for class 1**.**



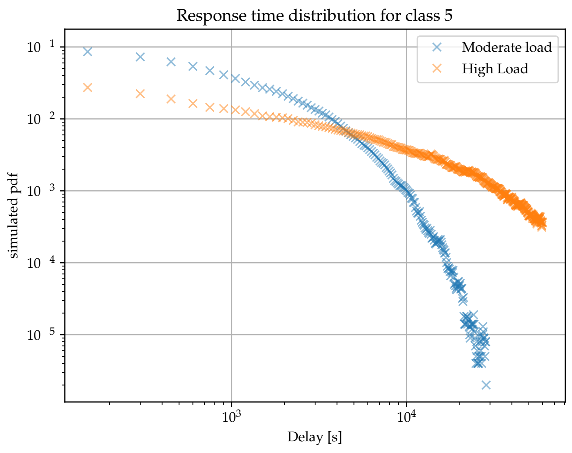
(c) Simulated response time distribution for class 2**.**



(d) Simulated response time distribution for class 3**.**

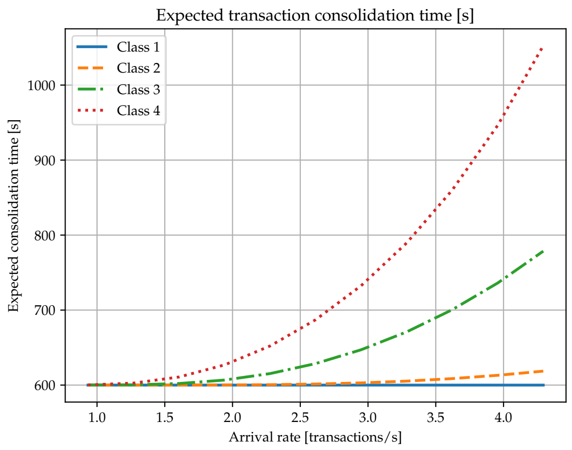


(e) Simulated response time distribution for class 4**.**

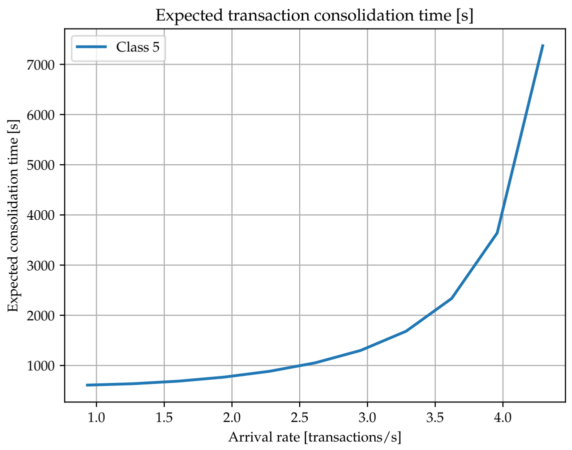


(f) Simulated response time distribution for class 5**.**

Figure 7: Distribution of the response time obtained with stochastic simulation in log-log scale. The parameters are shown in Table 1. The scheduling discipline within a class if FIFO.



(a) Expected consolidation time for classes $1 \ldots 4$ as function of the arrival rate.



(b) Expected consolidation time for classes 5 as function of the arrival rate.

Figure 8: Analysis of the expected consolidation time as function of the arrival intensity.

This emphasises the importance of the analytical model that provides accurate estimates in several orders of magnitude shorter times than the simulation.

7 CONCLUSION

In this paper, we have studied the relation between the arrival process of transactions at a blockchain distributed ledger and their expected consolidation delay. The consensus algorithm is the popular PoW. In our setting, transactions are grouped into classes that are ordered according to the offered consolidation fees.

A priority queueing model and its numerical solution have been proposed to evaluate the trade-off between the offered fees and the expected consolidation time. With respect to the state of the art, our approach does not require to run computationally expensive simulations which we have observed to converge very slowly. The efficiency of the solution method allows software systems that interact with blockchains to online evaluate the cheapest transaction fee that they have to offer to meet their non-functional service requirements.

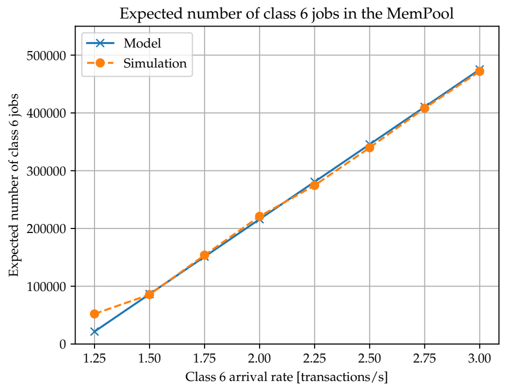
Another peculiarity of the model is that it takes into account two strategies of transaction dropping in case the arrival intensity exceeds the service capacity of the blockchain. The first is based on timeouts and the latter on a finite capacity buffer.

The model has been validated with measurements on the blockchain of Bitcoin and against several simulation experiments.

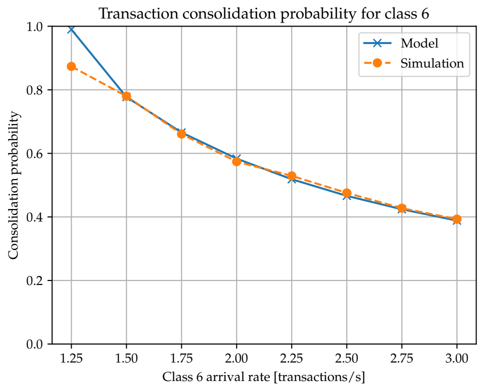
Future works include the analysis of the correlation between the arrival intensity of the transactions and the offered fees, and the study of optimality of the reneging policies.

REFERENCES

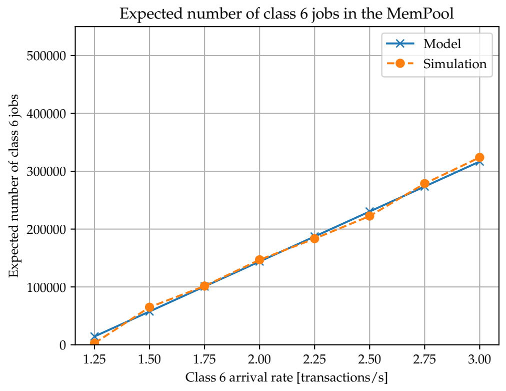
[1] M. Alharby and A. van Moorsel. 2018. The Impact of Profit Uncertainty on Miner Decisions in Blockchain Systems. *Electron. Notes Theor. Comput. Sci.* 340 (2018), 151–167.
[2] M. Alharby and A. van Moorsel. 2020. BlockSim: An Extensible Simulation Tool for Blockchain Systems. *Frontiers Blockchain* 3 (2020), 28.
[3] C. J Ancker and A. V. Gafarian. 1962. Queuing with impatient customers who leave at random. *J. of Industrial Engineering* 13 (1962), 84–90.
[4] Adam Back. 2002. *Hashcash - A Denial of Service Counter-Measure.* Technical Report. www.hashcash.org/papers/hashcash.pdf
[5] N.T.J. Bailey. 1954. On Queueing Processes with Bulk Service. *J. of the Royal Statistical Society, Series B* 16, 1 (1954), 80–87.
[6] S. Bocquet. 2005. *Queueing Theory with Reneging.* Technical Report. Defence Systems Analysis Division, Department of Defence, Australian Government.
[7] J. Buzen and A. Bondi. 1983. The response times of priority classes under preemptive resume in M/M/m queues. *Oper. Res.* 31, 3 (1983), 456–465.
[8] R. Davis. 1966. Waiting-time distribution of a multi-server, priority queuing system. *Oper. Res.* 14, 1 (1966), 133–136.
[9] V. De Maio, R. Brundo Uriarte, and I. Brandic. 2019. Energy and Profit-Aware Proof-of-Stake Offloading in Blockchain-based VANETs. In *Proc. of the 12th IEEE/ACM Int. Conf. on Utility and Cloud Computing, UCC.* 177–186.
[10] C. Decker and R. Wattenhofer. 2013. Information propagation in the bitcoin network. In *Proc. of the 13-th IEEE Int. Conf. on Peer-to-Peer Computing.* 1–10.
[11] I. Eyal and E. Günter Sirer. 2018. Majority is not enough: bitcoin mining is vulnerable. *Commun. ACM* 61, 7 (2018), 95–102.
[12] R. Gardner, P. Reinecke, and K. Wolter. 2019. Performance of Tip Selection Schemes in DAG Blockchains. In *Proc. of Mathematical Research for Blockchain Economy, 1st Int. Conf., MARBLE.* 101–116.
[13] S. Kasahara and J. Kawahara. 2019. Effect of Bitcoin fee on transaction-confirmation process. *J. of Industrial & Management Optimization* 15, 1 (2019), 365–386.
[14] Y. Kawase and S. Kasahara. 2020. Priority queueing analysis of transaction-confirmation time for Bitcoin. *J. of Industrial & Management Optimization* 16, 3 (2020), 1077–1098.
[15] O. Kella and U. Yechiali. 1985. Waiting time in the non-preemptive priority M/M/c queue. *Stochastic Models* 1, 2 (1985), 257–262.
[16] I. Malakhov, A. Marin, A. Rossi, and D. Smuseva. 2020. Fair Work Distribution on Permissioned Blockchains: A Mobile Window Based Approach. In *Proc. of the 3rd Int. Conf. on Blockchain.* 1–6.
[17] D. Miller. 1981. Computation of Steady-State Probabilities for M/M/1 Priority Queues. *Oper. Res.* 29, 5 (1981), 945–958.
[18] S. Nakamoto. 2009. Bitcoin: A peer-to-peer electronic cash system. http://www.bitcoin.org/bitcoin.pdf
[19] Till Neudecker, Philipp Andelfinger, and Hannes Hartenstein. 2016. Timing Analysis for Inferring the Topology of the Bitcoin Peer-to-Peer Network. In *Proc. of the 13th IEEE Int. Conf. on Advanced and Trusted Computing (ATC).* 358–367.
[20] H. Takagi. 1991. *Queueing analysis: a foundation of performance evaluation.* North-Holland, Amsterdam.
[21] J. Wang, O. Baron, and A. Scheller-Wolf. 2015. M/M/c Queue with Two Priority Classes. *Oper. Res.* 63, 3 (2015), 733–749.
[22] S. M. Werner, P. J. Pritz, A. Zamyatin, and W. J. Knottenbelt. 2019. Uncle Traps: Harvesting Rewards in a Queue-based Ethereum Mining Pool. In *Proc. of the 12th EAI Int. Conf. on Perf. Eval. Methodologies and Tools, VALUETOOLS.* 127–134.



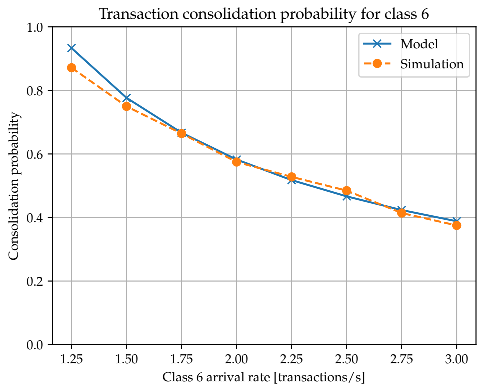
(a) Expected number of class 6 transactions in the MemPool. The expected reneging time is 72 hours.



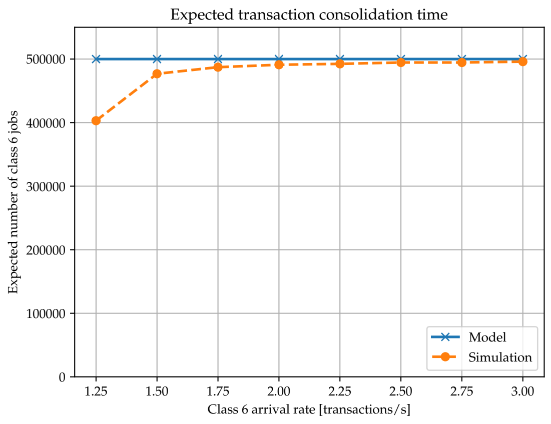
(b) Consolidation probability for class 6 transactions. The expected reneging time is 72 hours.



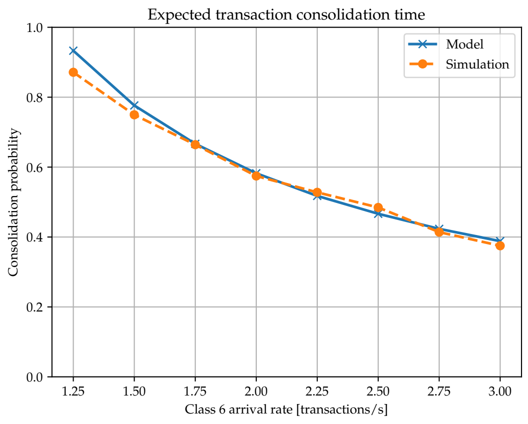
(c) Expected number of class 6 transactions in the MemPool. The expected reneging time is 48 hours.



(d) Consolidation probability for class 6 transactions. The expected reneging time is 48 hours.



(e) Expected number of class 6 transactions in the MemPool with buffer capacity for this class is $5 \cdot 10^5$.



(f) Consolidation probability for class 6 transactions. The buffer capacity for this class is $5 \cdot 10^5$.

Figure 9: Occupancy of the MemPool and probability of dropping in case of timeout and finite buffering.