# Performance of Bitcoin network with synchronizing nodes and a mix of regular and compact blocks

**Preprint** · August 2020

CITATIONS
0

READS
209

**3 authors:**

Jelena Mišić
Ryerson University
**431** PUBLICATIONS **4,268** CITATIONS
SEE PROFILE

Vojislav Misic
Ryerson University
**388** PUBLICATIONS **3,459** CITATIONS
SEE PROFILE

Xiaolin Chang
Beijing Jiaotong University
**129** PUBLICATIONS **744** CITATIONS
SEE PROFILE

# Performance of Bitcoin network with synchronizing nodes and a mix of regular and compact blocks

Jelena Mišić[1], Vojislav B. Mišić[1], and Xiaolin Chang[2]
[1]Ryerson University, Toronto, ON, Canada
[2]Beijing Key Laboratory of Security and Privacy in Intelligent Transportation
Beijing Jiaotong University, Beijing, China

***Abstract*—Compact blocks and compact block protocol are a recent addition to the Bitcoin (BTC) data propagation protocol that aims to reduce bandwidth requirements and, possibly, reduce latency as well. In this work we evaluate operation of BTC network under a mix of regular and compact block traffic, assuming that nodes randomly leave and re-join the network, and perform block and, optionally, transaction pool (mempool) synchronization upon returning. Our analysis begins by evaluating block and transaction deficits accumulated during the node absence and block synchronization. Then, we analyze mempool behavior and show that mempool synchronization is necessary since it decreases probability of transaction deficit and the need for transaction retrieval actions. Finally, we analyze the impact of synchronization activities and transaction deficit on data distribution times in the BTC network with high and low bandwidth distribution modes for a mix of compact and regular blocks. Results demonstrate resilience to node absence and subsequent synchronization, as well as substantial performance improvements for small protocol changes. Furthermore we show that the low bandwidth mode is more resilient to potential security attacks.**

***Index Terms*—Bitcoin; block and transaction propagation; compact blocks**

## I. Introduction

Excessive bandwidth usage is among the foremost networking problems that Bitcoin (BTC) community is trying to solve. Namely, the standard data relay protocol [3] uses a three-way handshake so the average latency includes the actual transmission time for the INV, GETDATA, and BLOCK messages, plus 1.5 round trip times (RTT) between the participating nodes. In reality, each of the messages is packaged into one or more TCP segments [10]. The first two messages are typically short and thus likely to fit within a single TCP segment, but the third one will require many TCP segments as block size is close to the current limit of 1MByte. Furthermore, the need to use unicast communications for data dissemination in the Bitcoin peer-to-peer (P2P) network causes many duplicate messages as transactions and blocks are advertised, and occasionally even transmitted, to peers that already have received and processed them. As the result, a lot of bandwidth is actually wasted.

To reduce bandwidth usage and/or latency, a number of additions to the basic functionality of Bitcoin were made over the years [2], most notably the following.

- *Unsolicited block push* allows a miner node to immediately send the newly mined block to its full node peers, as they cannot possibly know about it [3]. This reduces latency from 1.5RTT to only 0.5RTT, but bandwidth savings are not high as the omitted messages (INV and GETDATA) tend to be rather short in comparison with the BLOCK message. However, this option does not apply to relay blocks so the impact of those benefits is rather limited.
- *Direct header announcement* allows nodes to send the entire header of a new block instead of the header hash contained in an INV message. This allows the recipient node to partially verify the header before asking for the full block.
- *Compact blocks* contain only a subset of block transactions, presumably known to the recipient, sent in raw format while others are sent in compressed form [4]. As the result, the size of that message (CMPCTBLOCK) can be substantially reduced compared to the equivalent BLOCK message; for further savings, even the INV/GETDATA handshake can be omitted, similar to unsolicited block push described above.

The other source of excess bandwidth usage is the process of initial or re-synchronization of the local copy of the blockchain ledger by the nodes that join the Bitcoin network as well as for nodes that return to the network after a period of absence. Following the established practice in peer-to-peer networks, such absences are often referred to as node churn. To synchronize their ledgers, such nodes need to find out which blocks they are missing and then download them from one or more peers. In addition, nodes that want to take part in the mining process must find out which transactions have been propagated through the network but not yet included in accepted blocks, and download them from their peers so that their pools of unconfirmed transactions (mempools) are updated. Downloading of blocks and transactions puts an extra burden on the peers that supply them, essentially wasting their bandwidth at the expense of regular block and transaction relay.

Savings that can be obtained by using the compact block relay without churn have been investigated in [13]. Some experimental results regarding node churn have been presented in [8], [21], and a probabilistic analysis of node churn but with block synchronization only has been reported in [15]. However, an in-depth analysis of the impact of absence and subsequent re-synchronization and its performance in the network that uses both standard and compact block relay is still missing.

In this work we address the issue of node re-synchronization in the BTC network that relays both regular and compact blocks. When compact blocks are used, we consider the case

where mempools of returning nodes must be updated as well. We include models of high and low bandwidth modes for the distribution of the mix of standard and compact block traffic [4]. In high bandwidth mode we deploy three TCP connections as stipulated by the proposal [4] while other TCP connections carry regular blocks. For low bandwidth proposal we use three or six TCP connections while the rest of connections carry regular blocks.

Our main contributions are as follows.

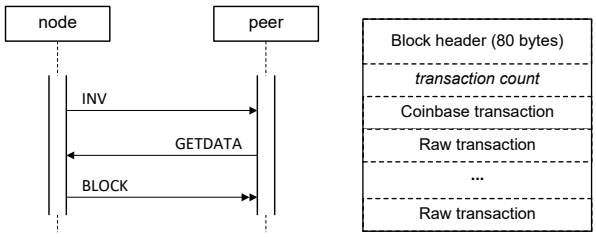
- We have developed a comprehensive probabilistic model of block and transaction synchronization phases, followed by an M/G/1 queuing model of node mempool which models the interplay of steady state, node absence and synchronization using vacations due to block inter-arrivals and node absence.
- We have modified the queuing network of the BTC network initially developed in [14] to include the impact of synchronization on the number of available TCP connections, the effective number of nodes that participate in regular block relay, and the mix of regular and compact blocks. For complete justification and model of connectivity, RTT and block size please refer to [14].
- We have performed a detailed performance evaluation of the BTC network in the scenarios with high bandwidth mode with three TCP connections, and low bandwidth mode with three and six TCP connections, both with and without mempool synchronization. Our analysis shows an improvement of block and transaction delivery times of around 13% and 35%, respectively, and a reduction of forking probability by about 22%. Using low bandwidth mode with six TCP connections for compact blocks improves previous results by additional 15%.
- Overall, the use of compact blocks shows noticeable performance improvements and demonstrates resilience to the additional load incurred by the re-synchronization process. Most notably, low bandwidth mode has potential for more bandwidth savings while keeping resiliency to double spending and variation of eclipsing attacks.

The rest of the paper is organized as follows. Section II explains the compact block relay protocol and block and transaction synchronization, while Section III presents the probabilistic models of block sizes, node's TCP connectivity and round trip times (RTTs). Section IV presents probabilistic model of block and transaction synchronization activities after returning after an absence. Section V presents a M/G/1 queuing model with vacations that represents steady state and synchronization for mempool. Section VI presents and discusses the results of performance evaluation obtained from two stages of the model: the first of mempool only, the second of the entire blockchain network. Finally, Section VII concludes the paper.

## II. Preliminaries

### A. Compact block relay

Standard BTC block relay includes exchange of three messages [3]. The new block is first announced with an INV message sent by the relay node; the recipient node will ask for the unknown block by responding with a GETDATA message.




(a) Low bandwidth relay handshake. (b) BLOCK message.

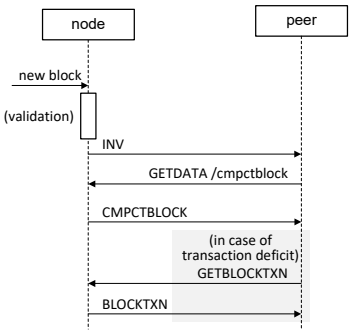
Fig. 1. Standard block relay.

The relay then sends the BLOCK message containing the entire block: its header as well as all the transactions in full – the so-called raw format. This exchange is schematically shown in Figs. 1(a) while the format of the BLOCK message is shown in Fig. 1(b).

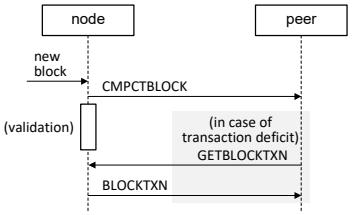
The compact block protocol has been proposed, together with a number of appropriate messages, to reduce bandwidth requirements [4]. Namely, mining of a block lasts much longer than mean transmission propagation time and it is safe to assume that most, if not all, of the transactions contained in the newly mined block have already propagated throughout the network at the moment the block is finally mined. Therefore, those transactions need not be sent with the block itself; instead, it suffices to send some short piece of information, referred to as 'short transaction IDs,' that will uniquely identify the transaction in question. Short transaction IDs are obtained by hashing the transaction ID and the beginning of the block header hash, and then dropping the two most significant bytes from the result to obtain a six-byte value. As the standard transaction hash is 32 bytes and a block can have more than 2,000 transactions, savings can be substantial.

Relaying of compact blocks can be achieved in one of two modes. In the so-called low bandwidth mode, relay node announces a new block with a regular INV or HEADERS message, and compact block is sent only if the peer responds with a GETDATA message with the appropriate option set, as shown in Fig. 2(a). The delay allows the relay node to perform block validation before forwarding. Low bandwidth mode can be performed on any number of TCP connections from a node without any restrictions on RTT towards the peers. In the so-called high bandwidth mode, the compact block is sent without prior announcement, similar to the unsolicited block push, perhaps even before its validation is completed, as shown in Fig. 2(b). On account of this, the original proposal recommends that the high bandwidth mode is used with no more than three peers [4].

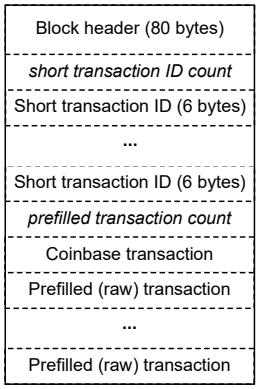
Not all transactions in a given block will be known to all peers in the network. At the very minimum, the coinbase transaction containing the miner information and related fees cannot be known before the block is actually mined and sent out. A few other transactions may not have propagated either. To account for this, the compact block allows a number of transactions, referred to as prefilled transactions, to be included in their original form similar to the standard BLOCK message, but with extra bytes that indicate the size of the




(a) Handshake, low bandwidth mode.




(b) Handshake, high bandwidth mode.




(c) CMPCTBLOCK message.

Fig. 2. Compact block relay (after [4]).

transaction. The format of the resulting message, referred to as CMPCTBLOCK, is shown in Fig. 2(c).

The recipient may still miss some of the transactions identified through their short IDs as the relay node cannot know which transactions the recipient does not have. This scenario is referred to as compact block transaction deficit and it is resolved by explicitly asking the relay node for missing transactions using a GETBLOCKTXN message that lists the corresponding short IDs. The relay node responds with a BLOCKTXN message that contains those transactions in full. In both messages, the order of transactions should match the order of their respective short IDs in the CMPCTBLOCK message. The resulting exchange is schematically shown in Figs. 2(a) and 2(b) for the low- and high bandwidth mode, respectively. The exchange takes 0.5RTT and 1.5RTT for high and low bandwidth modes, respectively, if all transactions with short ID are known to the recipient peer, or 1.5 and 2.5RTT otherwise.

Either way, peers must explicitly request the use of compact block relay; those that do not, as well as those nodes with an older version of Bitcoin daemon that does not support this mode, will still relay blocks using regular BLOCK messages [4]. The proposal recommends that high bandwidth mode is used with up to three peer nodes, preferably those with shortest measured RTT [4], but otherwise does not specify how to decide whether to use compact blocks at all.

### *B. Synchronization upon joining or rejoining the network*

All nodes should maintain an up-to-date blockchain ledger in order to validate and relay blocks. A node that joins the network for the first time, or rejoins after a period of absence, must synchronize its blockchain first. To this end, the returning node must ask a peer, referred to as the sync node, for headers of blocks that the node is missing using the GETBLOCKS message. The list begins with the last confirmed block or, in case of joining the network for the first time, with the genesis block itself. The sync node will respond with one or more INV messages that contain the headers of those blocks in reverse order, i.e., oldest blocks first. The returning node then sends GETDATA messages to the sync node in order to obtain the missing blocks. This approach is known as Blocks-First.

In the more recent Headers-First approach, the synchronization process is accelerated by using GETHEADERS and HEADERS messages to obtain block headers, and by requesting and downloading actual blocks from up to eight of the returning node's peers (possibly including the sync node) in parallel.

However, block synchronization does not include transactions which have not been packaged into a block yet. As the result, the pool of non-confirmed transactions (mempool) of the returning node will not include all transactions propagated during its absence, and the node cannot begin mining despite synchronizing its blockchain ledger. To synchronize its mempool, the returning node must request a list of transactions from a peer using the MEMPOOL message; the peer will then send the hashes of the transactions in its mempool using one or more INV messages. The returning node can then request full transactions it is missing using GETDATA message or several of them. The peer will send them using TX messages. Although transaction sizes are small, there can be many of them missing, and each requires a separate TX message. Note that there's no intrinsic order for transactions in the mempool, unlike blocks which are linked into the blockchain.

## III. Modeling block size and TCP connectivity

Let us assume that each compact block contains $M_l$ regular (i.e., prefilled) transactions and $M_s$ transactions identified with their respective short IDs, for a total of $M = M_l + M_s$. The sizes of Bitcoin message header, block header, and short transaction ID are 24, 80, and 6 bytes, respectively. Let $\overline{\Theta}$ denote the average size of a transaction in bytes. Then, the size of a CMPCTBLOCK message will be

$$B_c = \overline{\Theta} M_l + 6M_s + 80 + 24 \tag{1}$$

Regular BLOCK messages contain complete transactions only. According to the tracking web sites[1], block size ranges between 0.7MB and 1.25MB. Assuming mean transaction size of $\overline{\Theta} = 270$ bytes, block size expressed in number of transactions can be described with the probability generating function (PGF) of

$$B^*_{tr}(s) = \frac{e^{-2518s} + 2e^{-2700s} + e^{-3074s} + 2e^{-3222s} + 7e^{-3330s}}{13} \tag{2}$$

Assuming a single TCP connection has the throughput of 2Mbps [6] and that transmission time for regular and CMPCTBLOCK message with $M_s = 2500$ short transaction IDs

[1]https://bitinfocharts.com/bitcoin/ and https://www.blockchain.com/en/charts/avg-block-size, last accessed June 27, 2020.

are given by (1) and (2), respectively, we can obtain Laplace-Stieltjes transforms (LSTs) for regular and compact BLOCK message transmission times, expressed in seconds, as

$$B_r^*(s) = \frac{e^{-2.8s}+2e^{-3s}+e^{-3.4s}+2e^{-3.6s}+7e^{-4.4s}}{13} \quad (3)$$

$$B_c^*(s) = \frac{e^{-0.08s}+2e^{-0.32s}+e^{-0.68s}+2e^{-0.84s}+7e^{-0.96s}}{13} \quad (4)$$

with mean values equal to $b_r = -B_r^{`*}(0)$ and $b_c = -B_c^{`*}(0)$ respectively.

Regarding connectivity, we may distinguish between two large categories of nodes in the Bitcoin P2P network, depending on the number of TCP connections [14]. *Ordinary nodes* have fewer TCP connections and their probability distribution corresponds to a truncated binomial distribution with the PGF of $Cn(z) = \sum_{k=5}^{13} p_k z^k$ and mean value of about 8 [5], [12], [18]. *Gateway nodes* have a larger number of TCP connections which can be modeled using a long-tail distribution [11] with much higher node degrees. PGF which describes their connectivity distribution is

$$Lt(z) = L \sum_{k=14}^{60} \frac{1}{k^\kappa} z^k \quad (5)$$

where the shape and scaling factors are $2 \leq \kappa \leq 2.4$ and $L = 1/\sum_{k=14}^{60} \frac{1}{k^\kappa}$, respectively.

Following [5], [11], [12], [19] we assume that final connectivity distribution can be obtained by combining those two distributions as

$$Mx(z) = k_n Lt(z) + (1-k_n)Cn(z) = \sum_{i=n_{min}}^{i=n_{max}} mx_i z^i \quad (6)$$

where $n_{min} = 5$ and $n_{max} = 60$. Coefficient $k_n$ is in the range $0.3 \leq k_n \leq 0.7$. Mean number of connections per node is therefore obtained as $\overline{Mx} = Mx'(1)$.

Exchange of INV, GETDATA, and BLOCK messages takes 1.5RTT plus the actual message transmission times, for which the appropriate LST of $L^*_{1.5RTT}(s)$ (for the entire network coverage) has been derived in [14]. Round trip time of a TCP connection depends on physical distances between peers and delays through the routers. In the original BTC implementation, peer nodes can be located anywhere in the world [1], [6]. However, the compact block proposal [4] recommends that high bandwidth mode is used to exchange compact blocks with peers where RTTs are small. For that reason, we assume that peers that communicate using compact blocks are placed in proximity to the sender node, which means they are on the same continent. RTT within a given continent, say, North America, is obtained in a similar fashion by considering RTTs limited to 100ms using probability density function (pdf) and its LST, respectively, as

$$fc_{RTT}(x) = 0.3 C_f \mu_{NA} e^{-\mu_{NA}(x-0.01)} H(0.1-x)H(x-0.01)$$

$$L^*c_{1.5RTT}(s) = \int_{x=0.015}^{0.15} fc_{1.5RTT}(x) e^{-sx} dx \quad (7)$$

where $C_f$ is derived from the condition of total probability, while $H(x)$ denotes the Heaviside function with the values of 1, for $x \geq 0$, and zero elsewhere. In high bandwidth mode only 0.5RTT is needed, and the corresponding LST $L^*_{c,0.5RTT}(s)$ can be derived analogously to (7).

## IV. Modeling node absence and synchronization

We assume that both compact and regular blocks arrive to a node with exponentially distributed inter-arrival times at a rate of $\lambda_b = 1/600$ blocks per second. Let $k_r$ denote the proportion of regular blocks so that the arrival rates of regular and compact blocks are $\lambda_{br} = k_r \lambda_b$ and $\lambda_{bc} = \lambda_b(1-k_r)$, respectively. LST of block interarrival time is $V_b^*(s) = \frac{\lambda_b}{\lambda_b + s}$. Transaction arrival rate to the whole network is denoted with $\lambda_t$. Consequently, PGF for the number of transaction arrivals to the mempool during compact block inter-arrival time is [20], [9]:

$$F(z) = V_b^*(\lambda_t - z\lambda_t) = \sum_{k=0}^{\infty} f_k z^k \quad (8)$$

We assume that nodes are absent from the network up to once over a period of 24 hours, and that the duration of the absence period has an arbitrary probability distribution with LST $T^*_{off}(s)$ and mean $\overline{T_{off}} = -T^{*'}_{off}(0)$. Let $T_{day}$ denote the period of 24 hours expressed in seconds; then, the period when the node is present and active in the network (i.e., the active period) can be described with LST of $T^*_{act}(s) = e^{-sT_{day}}/T^*_{off}(s)$ with mean of $\overline{T_{act}} = T_{day} - \overline{T_{off}}$.

Upon returning to the network, the node needs to retrieve all blocks distributed in the network during its absence. To this end, it will establish TCP connections to up to eight peers and ask for the headers of the blocks it is missing. The number of such blocks can be described with the PGF of

$$N_b(z) = T^*_{off}(\lambda_b - z\lambda_b) = \sum_{k=0}^{k_{max}} n_k z^k \quad (9)$$

where limit $k_{max}$ can be set to a reasonably high multiple, say five to eight, of mean number of block arrivals during the period of absence. Assuming Headers-First approach is used, headers of the missing blocks will arrive in a single message as long as the backlog is smaller than 2000 blocks. This appears to be a reasonable limit for the block backlog if the period of absence is limited to one day, although extension to multiple HEADERS messages is straightforward. Since each block header has 80 bytes and throughput of TCP connection towards the peer which supplies the headers is 2Mbps, the LST for the time (in seconds) needed to download headers has the form

$$H^*(s) = L^*_{RTT}(s) N_b(e^{-s \cdot 0.00024}) \quad (10)$$

Upon getting the headers, node will attempt to get full blocks in parallel from up to eight directly connected peers. For simplicity, we assume the number of such peers is exactly eight (although random probability distribution of the number of peers can be readily introduced) and that each peer is asked to provide the same number of blocks which can be described with the PGF of

$$N_{bd}(z) = \sum_{k=0}^{k_{max}} n_k z^{\lceil k/8 \rceil} \quad (11)$$

Since download of each block requires RTT and block transmission time, LST for the full block download time becomes

$$T^*_{dld}(s) = N_{bd}\left(L^*_{RTT}(s)B^*_r(s)\right) \tag{12}$$

Then, total time to achieve block synchronization is

$$T^*_{sync}(s) = H^*(s)T^*_{dld}(s) \tag{13}$$

Some of the transactions in the compact block referenced via their short IDs may be absent from the node's mempool, thus creating a compact block transaction deficit. Regular blocks don't create this deficit since they contain complete transactions. Transaction deficit may occur during active state due to finite propagation time of transactions, but is much more pronounced during the synchronization process. If newly downloaded blocks deliver all transactions generated during the absence period, a deficit can only occur during retrieval of block headers and block download. We will refer to this as synchronization deficit $\delta(z)$, and its PGF can be derived from (13) as $\delta(z) = T^*_{sync}(\lambda_t - z\lambda_t)$.

Transaction deficit, denoted by the PGF of $\beta(z)$, occurs if downloaded blocks do not contain all transactions that arrived during the period of absence. This distribution is virtually impossible to estimate, hence we adopt the approximation $\beta(z) \approx \delta(z)$. The PGF for the total number of missing transactions after the synchronization process is completed is

$$\alpha(z) = \delta(z)\beta(z) \approx \delta(z)^2 \tag{14}$$

The returning node may decide to retrieve only missing blocks but not missing transactions, which will result in more interactions with its peers upon future block arrivals and, ultimately, slow down node operation. The synchronization process may be completed by requesting missing transactions using MEMPOOL message(s), which lasts $\overline{T_{memp}} = 2\overline{L_{RTT}} + \overline{\alpha}(8\overline{\Theta})/(2 \cdot 10^6)$ seconds.

In the analysis that follows, we will evaluate system performance with and without transaction synchronization.

## V. Queuing model of the node mempool

Assuming that a node is absent from the network with mean absence period of $\overline{T_{off}}$ per 24 hours, the probability that the node is active (inactive) is $\xi = 1 - \dfrac{\overline{T_{off}}}{T_{day}}$ ($\dfrac{\overline{T_{off}}}{T_{day}} = 1 - \xi$), respectively. If compact blocks are used, we assume that the relay node sends most recent received transactions as prefilled and older ones in the short form, so as to increase the probability of finding the latter transactions in the recipient's mempool.

To model the transaction deficit, let us consider the mempool as a queue which is serviced by up to $M$ transactions at a time. Namely, a regular block contains $M$ full transactions which are temporarily added to mempool and subsequently removed from it after validation. A compact block has $M_l$ prefilled transactions which are processed in the same way; it also has $M_s$ transaction identified by their short IDs which need to be verified against the mempool. Now, if the queue has $M$ or more transactions, we assume that the block can be validated and transactions can be removed from the mempool. However, if the queue has less than $M$ transactions at the arrival of a compact block, the node cannot validate all transactions (it knows only their short ID hashes). It then needs to retrieve their full versions from the sender of the compact block which creates additional traffic towards the peer that diminishes the bandwidth savings offered by the compact block transmission.

To model the transaction pool, we use an M/G/1 system with vacations and batch service. In this system, block inter-arrival time is a vacation which occurs in regular working regime as the result of relaying or mining by the target node. In addition, the absence of a node is a vacation too. The timing for this scenario is schematically presented in Fig. 3. Initially, we assume a constant number of transactions and constant transaction size in the block; random block size will be considered in the following Section. PGFs for the number of transaction arrivals during regular and compact block service times in high- and low bandwidth modes, respectively, given in (3), are:

$$A(z) = B^*_r(\lambda_t - z\lambda_t)L^*_{1.5RTT}(\lambda_t - z\lambda_t) = \sum_{k=0}^{\infty} a_k z^k \tag{15}$$

$$\begin{aligned} U_h(z) = {} & B^*_c(\lambda_t - z\lambda_t)\left((1-P_d)L^*_{c0.5RTT}(\lambda_t - z\lambda_t)\right. \\ & \left. + P_d L^*_{c1.5RTT}(\lambda_t - z\lambda_t)\right) = \sum_{k=0}^{\infty} u_{h,k} z^k \end{aligned} \tag{16}$$

$$\begin{aligned} U_l(z) = {} & B^*_c(\lambda_t - z\lambda_t)L^*_{1.5RTT}(\lambda_t - z\lambda_t)\cdot \\ & \left((1-P_d) + P_d L^*_{RTT}(\lambda_t - z\lambda_t)\right) = \sum_{k=0}^{\infty} u_{l,k} z^k \end{aligned} \tag{17}$$

For simplicity, we denote PGF for arrival process during compact block service only as $U(z) = \sum_{k=0}^{\infty} u_k z^k$ where $u_k = u_{l,k}$ or $u_k = u_{h,k}$, depending on the relaying mode.

### A. Markov points

We also need to choose Markov points which are characterized by the property that the number of transactions in the mempool at the current Markov point depends only on the previous Markov point and transaction arrivals between the two [20]. In this model, there are three types of Markov points, as shown in Fig. 3:

1) Moments of returning from block inter-arrival vacations, at which time the mempool contains $k$ transactions with the probability $q_k$.
2) Moments of block departures from the node, at which time the mempool contains $k$ transactions with the probability $h_k$.
3) Moments of returning from an absence (sleep) vacation (including block and transaction synchronization), at which moment the mempool contains $k$ transactions with the probability $s_k$.

Let $\xi = 1 - \dfrac{\overline{T_{off}}}{T_{day}}$ denote probability that a vacation is just a block vacation in steady state; the probability that it is an

absence vacation is, then, $1-\xi$. Equations that model the number of transactions in the mempool at Markov points are:

$$
\begin{aligned}
s_k &= (1-\xi)\sum_{j=0}^{k} h_j \alpha_{k-j} \\
q_k &= \xi \sum_{j=0}^{k} h_j f_{k-j} + \sum_{j=0}^{k} s_j f_{k-j} \qquad (18)\\
h_k &= \begin{cases} \sum_{j=0}^{M-1} q_j \left(k_r a_k + (1-k_r)u_k\right), & j < M \\ \sum_{j=M}^{M+k} q_j \left(k_r a_{k-j+M} + (1-k_r)u_{k-j+M}\right), & j \geq M \end{cases}
\end{aligned}
$$

This system can be reduced to only two equations as the first equation (the one that models the number of transactions in the mempool upon return from synchronization) can be eliminated:

$$
\begin{aligned}
q_k &= \xi \sum_{j=0}^{k} h_j f_{k-j} + (1-\xi)\sum_{j=0}^{k} h_j \sum_{i=0}^{j} \alpha_{k-i-j} f_i \qquad (19)\\
h_k &= \begin{cases} \sum_{j=0}^{M-1} q_j \left(k_r a_k + (1-k_r)u_k\right), & j < M \\ \sum_{j=M}^{M+k} q_j \left(k_r a_{k-j+M} + (1-k_r)u_{k-j+M}\right), & j \geq M \end{cases}
\end{aligned}
$$

The last system can be used to form PGFs of the state of mempool in two Markov points as $Q(z) = \sum_{k=0}^{\infty} z^k q_k$ and $H(z) = \sum_{k=0}^{\infty} z^k h_k$, respectively. After multiplication of left- and right-hand sides of the equations with $z^k$ and summation over $k = 0 \ldots \infty$, we obtain

$$
\begin{aligned}
Q(z) &= \xi \sum_{k=0}^{\infty} z^k \sum_{j=0}^{k} h_j f_{k-j} \\
&\quad + (1-\xi)\sum_{k=0}^{\infty} z^k \sum_{j=0}^{k} h_j \sum_{i=0}^{j} \alpha_{k-i-j} f_i \qquad (20)\\
H(z) &= \sum_{k=0}^{\infty} z^k \sum_{j=0}^{M-1} q_j \left(k_r a_k + (1-k_r)u_k\right) \qquad (21)\\
&\quad + \sum_{k=0}^{\infty} z^k \sum_{j=M}^{M+k} q_j \left(k_r a_{k-j+M} + (1-k_r)u_{k-j+M}\right)
\end{aligned}
$$

After changing the order of indicated summations and completing them, we obtain the PGFs as

$$
\begin{aligned}
Q(z) &= H(z)F(z)\left(\xi + (1-\xi)\alpha(z)\right) \qquad (22)\\
H(z) &= Au(z)\left(Q_M(1) + \frac{Q(z) - Q_M(z)}{z^M}\right) \qquad (23)
\end{aligned}
$$

where $Q_M(z) = \sum_{k=0}^{M-1} q_k z^k$, and $Au(z) = k_r A(z) + (1-k_r)U(z)$ denotes the PGF for the composite transaction arrival process during block service time; and $\overline{Au} = Au'(1)$.

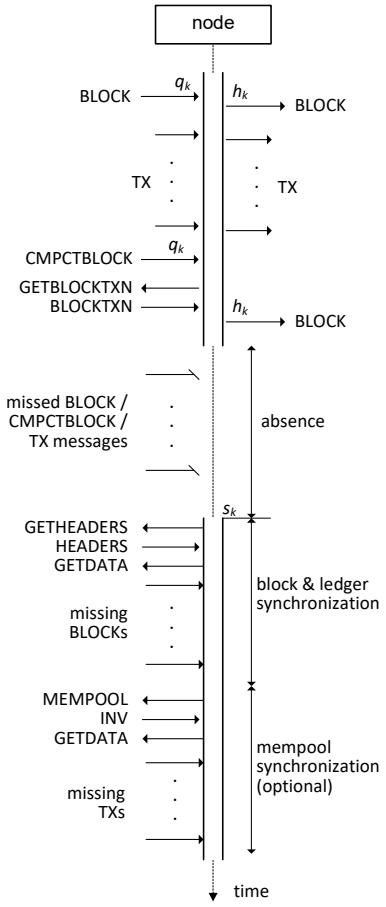



Fig. 3. Pertaining to the M/G/1 model of transaction memory pool (mempool).

We can further decouple $Q(z)$ and $H(z)$ as

$$
Q(z) = \frac{Au(z)F(z)\left(\xi + (1-\xi)\alpha(z)\right)\left(z^M Q_M(1) - Q_M(z)\right)}{z^M - Au(z)F(z)\left(\xi + (1-\xi)\alpha(z)\right)} \qquad (24)
$$

$$
H(z) = \frac{Au(z)\left(z^M Q_M(1) - Q_M(z)\right)}{z^M - Au(z)F(z)\left(\xi + (1-\xi)\alpha(z)\right)} \qquad (25)
$$

Function $Q_M(z)$ can be found from the normalization condition that $Q(1) + H(1) = 1$. By using L'Hôpital's rule we get:

$$
\begin{aligned}
Q(1) &= \frac{MQ_M(1) - Q'_M(1)}{M - \overline{Au} - \frac{\lambda_t}{\lambda_b} - (1-\xi)\overline{\alpha}} \\
H(1) &= Q(1) \qquad (26)
\end{aligned}
$$

The last equation can be represented as a differential equation:

$$
MQ_M(1) - Q'_M(1) = 0.5(M - \overline{Au} - \frac{\lambda_t}{\lambda_b} - (1-\xi)\overline{\alpha}) \qquad (27)
$$

which can be solved by substitution $Q_M(z) = e^{wz}$:

$$
Me^w - we^w = 0.5(M - \overline{Au} - \frac{\lambda_t}{\lambda_b} - (1-\xi)\overline{\alpha}) \qquad (28)
$$

Solving the previous equations allows one to obtain complete PGFs $Q(z)/Q(1)$ and $H(z)/H(1)$.

### B. Computing performance descriptors

Results from the previous Section hold when the block size is constant in the number of transactions. However, real block sizes are following a random distribution as shown in (2). For compact blocks, we assume that randomness in block size is coming from the random number of short transactions while the number of prefilled transactions is constant. In this case we need to compute a series of PGFs $Q_i(z)$ and $H_i(z)$ for each block size from the distribution, and compute total PGFs as

$$Q_{tot}(z) = \sum_{i=1}^{i_{max}} p_i Q_i(z) / (\sum_{i=1}^{i_{max}} p_i Q_i(1)) \tag{29}$$

$$H_{tot}(z) = \sum_{i=1}^{i_{max}} p_i H_i(z) / (\sum_{i=1}^{i_{max}} p_i H_i(1)) \tag{30}$$

Mean number and standard deviation of number of transactions in the mempool upon arrival of block can be obtained as

$$\overline{Q_{tot}} = Q'_{tot}(1) \tag{31}$$

$$\sigma(Q_{tot}) = \sqrt{Q''_{tot}(1) - \overline{Q_{tot}}^2 + \overline{Q_{tot}}} \tag{32}$$

The receiving node will experience transaction deficit if some of the transactions identified through their short IDs are not in the mempool; they must be obtained from the peer with an extra handshake. Assuming random block size, probability of transaction deficit for compact blocks is

$$P_d = \sum_{i=1}^{i_{max}} p_i \sum_{j=0}^{M_{s,i}} q_{i,j} \tag{33}$$

where $Q_i(z) = \sum_{j=0}^{\infty} q_{i,j} z^j$ and mean number of retrievals of deficit transactions upon a compact block arrival during the active time of the node is

$$\overline{N_{dt}} = P_d \lambda_{bc} \overline{T_{act}} \tag{34}$$

### C. Impact on TCP connectivity

TCP connection towards a peer may be busy due to the following activities.

1) Block header download after the node absence period, as per (10).
2) Missing block download, as per (12).
3) Retrieving missing transactions that arrived during block synchronization or during node absence, but were not included in blocks; the number of such transactions is given in (14).
4) Retrieving deficit transactions that did not exist in the mempool when a compact block arrived in steady state; the number of such transactions is given in (34).

Mean value of the total time that the node is involved in the activities listed above is $\overline{T_{oh}} = \overline{H} + \overline{T_{dld}} + \overline{T_{memp}} + \overline{N_{dt}} RTT$. This has to be mapped to probability that TCP connection between ordinary node and its peer is not available for regular transaction and block traffic. However proposal [4] recommends that the returning node synchronizes with more than one peer for reliability and security reasons. This impacts the probability that the TCP connection is unavailable for regular block and transaction traffic so it is in the range $\frac{\overline{T_{oh}}}{\overline{Cn}\overline{T_{act}}} \leq P_{tcp} \leq \frac{\overline{T_{oh}}}{\overline{T_{act}}}$.

PGFs for TCP connectivity in presence of absence, synchronization and update have to be modified from the values $Cn(z)$ and $Lt(z)$ (presented in Section III) as follows:

$$Cn_m(z) = \xi \left( \sum_{i=0}^{\overline{Cn}} \binom{\overline{Cn}}{i} P_{tcp}^i (1 - P_{tcp})^{(\overline{Cn} - i)} \frac{Cn(z)}{z^i} \right) + (1 - \xi) \tag{35}$$

$$Lt_m(z) = \sum_{i=0}^{\overline{Lt}} \binom{\overline{Lt}}{i} P_{tcp}^i (1 - P_{tcp})^{(\overline{Lt} - i)} \frac{Lt(z)}{z^i} \tag{36}$$

where polynomials $Cn(z)/z^i$ and $Lt(z)/z^i$ contain positive integer powers of variable $z$ obtained by rounding non-integer exponents to closest lower value. These PGFs are further used to form PGF for overall connectivity and effective number of nodes, respectively, as

$$Mx_m(z) = k_n Lt_m(z) + (1 - k_n) Cn_m(z) \tag{37}$$

$$N_{eff}(\xi) = k_n N + (1 - k_n) \xi N \tag{38}$$

which are used in the blockchain queuing model.

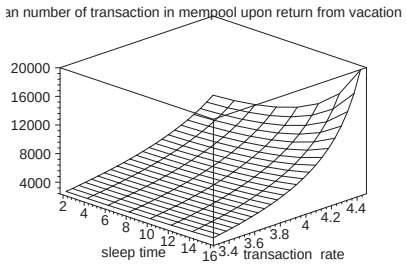
## VI. Performance evaluation

To evaluate the performance of the network with block and, optionally, mempool synchronization in the presence of block traffic consisting of regular and compact blocks, we have solved the system of equations presented above. Absence period was assumed to follow Erlang-$k$ distribution with $k = 2$ and its mean value was varied from 2 to 16 hours. Mean transaction size was $\overline{\Theta} = 270$ bytes, while the block size distribution was set as per (2). Total number of short transactions in a compact block was set to $M_s = 2500$ while the remaining ones were prefilled. Regular blocks had $M_l = 2500$ raw transactions. We have evaluated both high bandwidth mode with three TCP connections carrying carried compact blocks, and low bandwidth mode with three and six TCP connections carrying compact blocks, respectively.

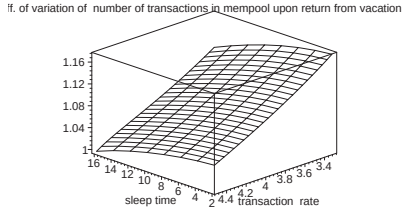
### A. Mempool evaluation

In the first experiment, we have considered scenarios with and without mempool synchronization upon returning from absence and block synchronization. Total transaction deficit at the end of these synchronization phases was modeled as $\alpha(z) = \delta(z)^2$. In the second case only block synchronization was performed but without subsequent mempool synchronization. Transaction arrival rate for the entire network was varied between 3.2 to 4.5 transactions per second, which corresponds to the mean and peak daily average values from January 2019 to end of June 2020 provided on the tracking web sites listed above. As any given node has to eventually receive all the transactions injected into the network and all the blocks mined, size of the network does not matter in this case.

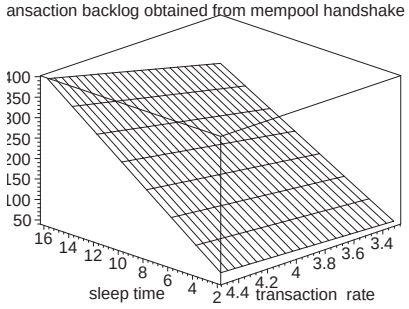
Performance descriptors for the case with mempool synchronization are shown in Fig. 4. Results are mostly affected



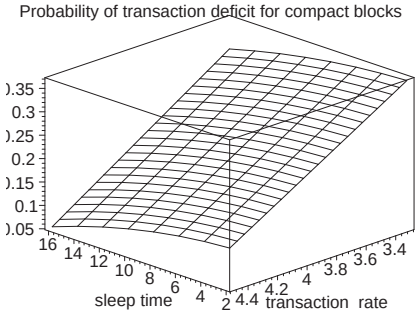

(a) Mean number of transactions in mempool upon return from vacation.



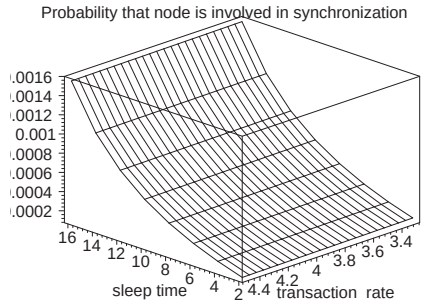

(b) Coefficient of variation of the number of transactions in mempool upon return from vacation.




(c) Transaction backlog obtained from mempool synchronization.




(d) Probability of transaction deficit (compact blocks only).




(e) Probability that node is busy in the synchronization process.

Fig. 4. Mempool performance with mempool synchronization.

with length of absence period; as the results for high and low bandwidth modes are very similar, we show only the former. Mean number of transactions in mempool upon return from absence, Fig. 4(a), including absence, block, and mempool synchronization, increases super-linearly with transaction rate and mean absence period; however, its coefficient of variation, Fig. 4(b), decreases. Values of the coefficient of variation larger than one correspond to small transaction arrival rates and short absences in which case sub-geometrically distributed block size strongly affects the distribution of a comparatively small number of transactions in the mempool. At higher transaction arrival rates and longer absence periods, mempool size is affected by the increased number of Poisson-distributed transactions arrivals and the coefficient of variation is closer to one.

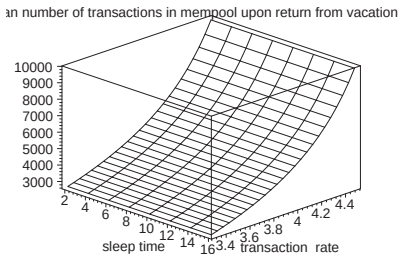
Mean transaction backlog, Fig. 4(c), increases almost linearly with the absence period, but shows a very mild dependence on transaction arrival rate due to the linear dependence of block synchronization time on these two parameters. Probability of transaction deficit for compact blocks, shown in Fig. 4(d), exhibits the behavior similar to that of the coefficient of variation for PGF $Q(z)$. It drops to 0.05 at large transaction arrival rates and large absence periods where mempool is saturated with transactions. For short absences of two hours and small transaction arrival rate of 3.2 transactions per second, the probability of deficit reaches 35%; at this point, there are (on the average) slightly more than 2700 transactions in the mempool upon block arrival.

Probability that node is active and busy with synchronizing its ledger and mempool is the ratio of total mean overhead time and mean active time; it increases with the absence period but is virtually unaffected by the transaction arrival rate, as can be seen from Fig. 4(e). This is caused by comparatively short absences which limit the number of blocks and transactions to retrieve during the synchronization period.

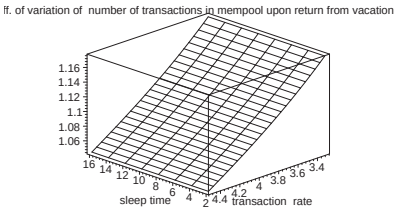
Performance descriptors for the case without mempool synchronization are shown in Fig. 5. As expected, the curves are independent of the value of mean absence period. At small transaction arrival rates, mean number of transactions in the mempool, its coefficient of variation, and probability of transaction deficit have similar values to those in case with mempool synchronization, which is due to the comparatively small number of transaction arrivals during block synchronization time. However, when the transaction arrival rate exceeds about 3.5 transactions per second, the differences becomes more pronounced. For example, probability of transaction deficit for compact blocks at mean absence of 16 hours and transaction arrival rate of 4.4 transactions per second is as high as 15%, compared to 5% for the case with mempool synchronization from Fig. 4(d). Probability that the node is active and busy in overhead activity, Fig. 5(d), is slightly lower than in the case with mempool synchronization; the price to pay is noticeably higher probability of transaction deficit, Fig. 5(c).

### *B. Connectivity and node/transaction traffic*

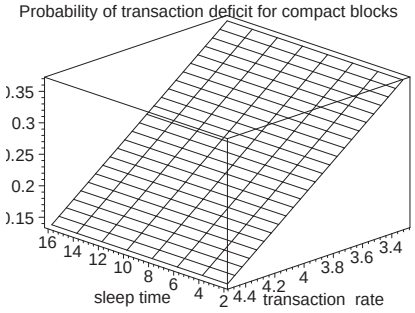
Connectivity of each node was modeled using PGF $Mx_m(z)$ defined in (37) with $k_n = 0.4$, while the long-tail parameter in (5) was set to $\kappa = 2$. Consequently, the effective number of



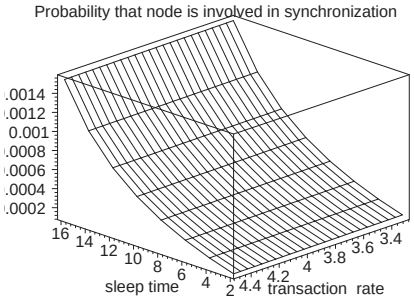

(a) Mean number of transactions in mempool upon return from vacation.




(b) Coefficient of variation of the number of transactions in mempool upon return from vacation.

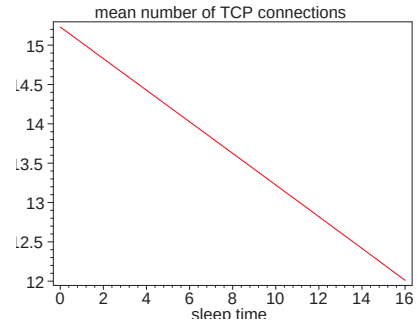



(c) Probability of transaction deficit (compact blocks only).

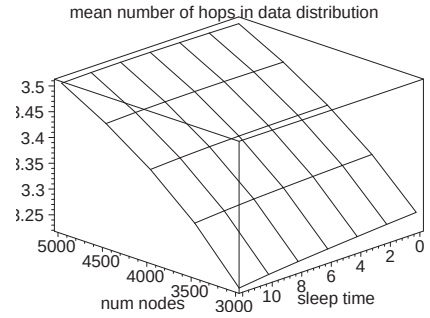



(d) Probability that node is busy in the synchronization process.

Fig. 5. Mempool performance for high bandwidth mode without mempool synchronization.




(a) Distribution of the number of connections per node vs. mean absence period.




(b) Mean number of hops needed to reach the entire network.

Fig. 6. Connectivity and data distribution properties of the network.

nodes in the network was $N_{eff} = 0.4N + 0.6\xi N$. Transaction arrival rate for the entire network was set to $\lambda_t = 4.3$ transactions per second.

Each node in high bandwidth mode was receiving compact blocks over three TCP connections from the peers in its proximity and regular blocks over remaining connections, as recommended by [4]. In low bandwidth mode, we assumed that nodes receive compact blocks over $k_c = 3$ and 6 TCP connections, without requirements for peer proximity. Total LST for block service time can then be expressed for high and low bandwidth mode as

$$B^*_{h,mix}(s) = \frac{\overline{Mx_m} - 3}{\overline{Mx_m}} B^*_r(s) L^*_{c1.5RTT}(s) + \frac{3}{\overline{Mx_m}} B^*_c(s) \cdot ((1 - P_d) L^*_{c0.5RTT}(s) + P_d L^*_{c1.5RTT}(s)) \quad (39)$$

$$B^*_{l,mix}(s) = \frac{\overline{Mx_m} - k_c}{\overline{Mx_m}} B^*_r(s) L^*_{1.5RTT}(s) + \frac{k_c}{\overline{Mx_m}} B^*_c(s) \cdot L^*_{1.5RTT}(s) \left((1 - P_d) + P_d L^*_{RTT}(s)\right) \quad (40)$$

We have conservatively assumed that each node's TCP connection is equally contributing to the synchronization process upon returning from an absence so that $P_{tcp} = \frac{\overline{T_{oh}}}{\overline{T_{act}}}$.

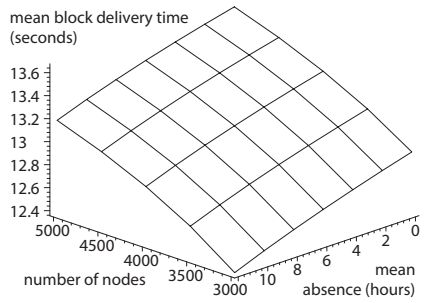
These changes were then inserted in the model for data propagation that consists of block and transaction distribution algorithm (which allows calculation of non-homogeneous Poisson block/transaction arrival rates for each node) and priority queuing model for blocks and transactions at each node [14]. In this way, we were able to calculate probability distributions of node response time for blocks and transactions, as well as their network distribution time.

Since results for high- and low bandwidth modes for connectivity are similar as they mostly depend on the absence period, we show only results for high bandwidth mode. Distribution of the number of connections per node, i.e., node connectivity, is shown in Fig. 6(a). We note that the number of usable connections declines by about 17% when mean absence time changes from 0 to 12 hours.

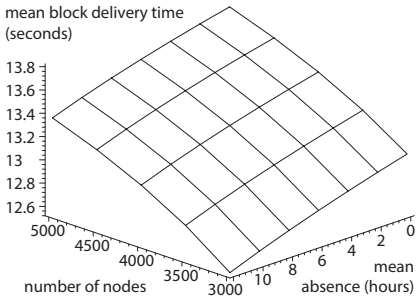
Fig. 6(b) shows mean number of hops needed to propagate a block through the network. It was computed using the algorithm from [14] but including the impact of connections that are rendered unavailable by the synchronization activities distribution algorithm. As expected, mean number of hops increases with network size; at smaller network sizes, it also decreases with the absence period as the number of operational nodes is effectively reduced by periodic absences. However for about 4000 nodes and above, the number of hops becomes virtually independent on the absence period on account of large number of nodes and rich TCP connectivity.

### C. Network delivery time

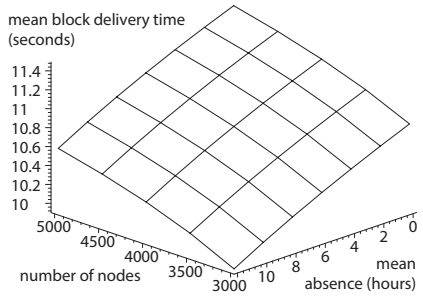
Network delivery time for blocks is shown in Fig. 7 for both high- and low bandwidth modes; in the latter case, we have presented results for compact blocks transmitted over 3 and 6 TCP connections, respectively. Since transaction arrival rate was set to 4.3 transactions per second for the whole network, probability of transaction deficiency, Fig. 4(d), was between $P_d = 0.05$ and $P_d = 0.12$. Transaction deficit requires an



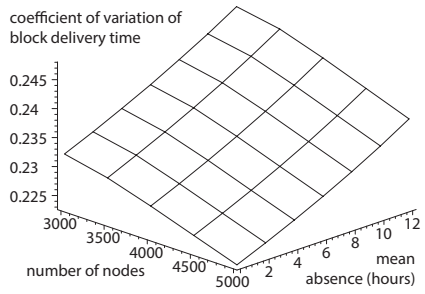

(a) Mean delivery time (seconds), 3 TCP connections for compact blocks in high bandwidth mode.



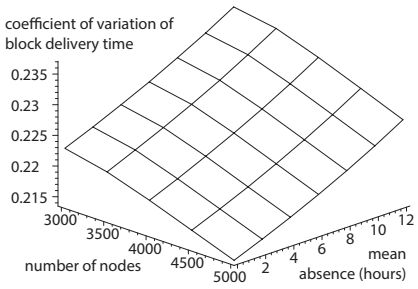

(b) Mean delivery time (seconds), 3 TCP connections for compact blocks in low bandwidth mode.



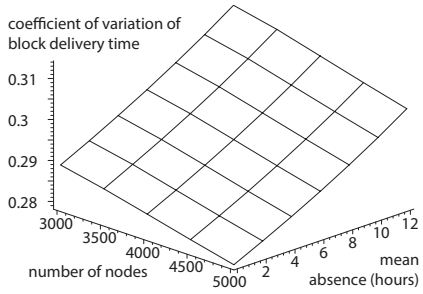

(c) Mean delivery time (seconds), 6 TCP connections for compact blocks in low bandwidth mode.




(d) Coefficient of variation of block delivery time, 3 TCP connections for compact blocks in high bandwidth mode.

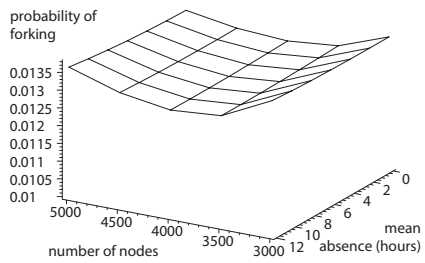



(e) Coefficient of variation of block delivery time, 3 TCP connections for compact blocks in low bandwidth mode.

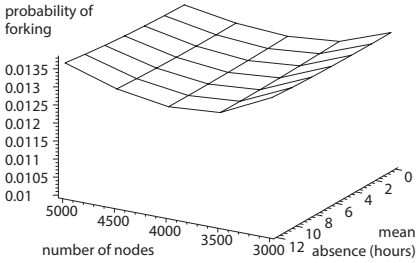



(f) Coefficient of variation of block delivery time, 6 TCP connections for compact blocks in low bandwidth mode.

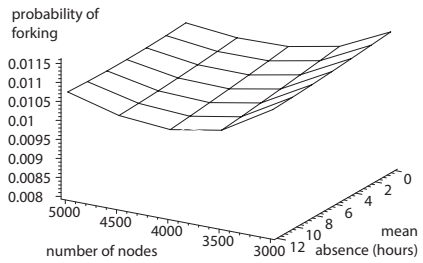
Fig. 7. Network delivery time for blocks.




(a) High bandwidth mode.




(b) Low bandwidth mode with compact blocks over 3 TCP connections.




(c) Low bandwidth mode with compact blocks over 6 TCP connections.

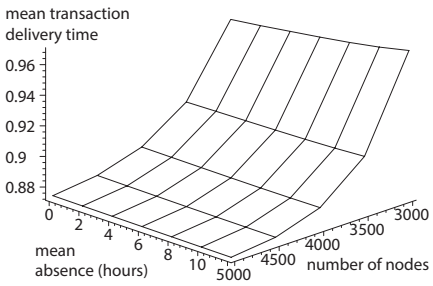
Fig. 8. Forking probability.

additional handshake between recipient and sender nodes in order to retrieve missing transactions in full form.

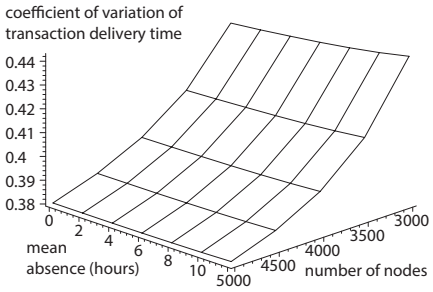
For all three cases, when the network size increases, mean block delivery time increases while its coefficient of variation decreases. Both trends are caused by the increase in number of hops for data distribution that, in turn, leads to an increase in the sum of node response times which are identically distributed random variables. On the other hand, mean block delivery time decreases with mean absence period due to the decrease of the number of active nodes and, consequently, reduced effective network size; however, the coefficient of variation increases. Results obtained without node absence are about 13% lower than those obtained without node absence but with regular blocks only [14].

When we compare three modes of compact block deployment, we see the combined impact of round trip times and number of TCP connections dedicated to compact block transmission. Namely, high and low bandwidth modes with 3 TCP connections differ only in that the former uses 0.5RTT towards nodes in proximity, compared to 1.5 of general RTT needed by the low bandwidth mode. As the result, mean delivery time in high bandwidth mode is smaller by only 2% compared to low bandwidth mode. The reason is that the majority of TCP connections still convey full blocks which require several seconds for transmission. However coefficient of variation is about 4% higher in high bandwidth mode due to impact of RTT. Increasing the number of TCP connections for compact blocks from 3 to 6 decreases block delivery time by around 15%, but this comes at the expense of an increase of coefficient of variation by around 22%.

A blockchain fork is the event when one or more newly




(a) Mean delivery time for transactions.




(b) Coefficient of variation of delivery time for transactions.

Fig. 9. Network delivery time for transactions, 3 TCP connections in high bandwidth mode.

mined blocks appear in the network while a previously mined block is still in the process of distribution. This scenario results in linking of different blocks at the tip of different node blockchains, thus leading to temporary inconsistent state of the ledger. Probability of a fork is very much dependent on block delivery time. As can be seen in Figs. 8(a) and 8(b), forking probability takes on values between 1.30% to 1.37% which are lower by about 22% than the corresponding value obtained when all blocks are distributed as regular blocks. The improvement is due to the reduction of the coefficient of variation of block distribution time; it increases with mean absence period via the increase of coefficient of variation of block delivery time. These two figures are very similar since there are opposite small changes in mean value and coefficient of variation of block delivery time.

The real winner is presented in Fig. 8(c) for compact block delivery over 6 TCP connections in low bandwidth mode. This mode brings an additional 10% of reduction of forking probability due to the combined effect of reduction of mean block distribution time and increase of coefficient of variation of block distribution time. We note that forking probability has a mild minimum at network size of about 4000 nodes, which is the consequence of the fact that the mean number of hops in data distribution grows with network size and increases the time window when forking can occur. On the other hand, the arrival rate of mined blocks per node increases with the decrease of network size which increases the likelihood of a new block arrival in small time window.

Network delivery time for transactions is shown in Fig. 9. We present only the case for high bandwidth mode since low bandwidth mode gives very similar results. Without node absence, mean transaction delivery time is about 35% lower than in the case with regular blocks only [14]. Similar observation holds for the coefficient of variation of transaction delivery time, but in this case values are smaller due to the summation of node transaction response times which are identically distributed random variables. Still, the coefficient of variation of delivery time for transactions is higher than the one for blocks since transactions are served from a lower priority queue [14].

### D. Security considerations

First, let us note that the decrease of forking probability diminishes the impact of attacks on ledger consistency and in that sense compact block technique is beneficial.

On the other hand, high bandwidth mode enables node clustering within communities which can augment problems caused by lack of block validation before forwarding. Namely, if the mining node creates a block with double spending transactions, this block can propagate quickly through the originating community and further through the whole network. While the malformed transaction(s) will eventually be found and removed, fast propagation of such blocks means that the time to detect and correct the problem will be longer than in the scenario with regular blocks only.

As the problem is mostly caused by the block being forwarded before being properly validated, the low bandwidth mode with the same number of TCP connections is much more resilient to such attack since it includes block validation before forwarding. Our results show that the difference in performance between high- and low bandwidth modes with the same number of TCP connections is only a few percent, which seems like a small price to pay for the increased resiliency to double spending attacks.

Another kind of attack is a variant of eclipsing attack initially introduced for regular blocks in [7], [16], [17]. In this case, the sender may enter into transaction deficit handshake with a recipient coming back from an absence, but then refuse to send full transactions when requested to do so. To combat this kind of attack, the node which has received a compact block with transaction deficit needs to set a time-out after which it will stop waiting for the reply and discard the compact block. As the low bandwidth mode does not give preference to nearby nodes when establishing TCP connections for compact blocks, it is more resilient to such an attack with compact blocks, as the eclipsing attack is harder to launch using geographically distant nodes administered by different authorities.

## VII. Conclusions

In this work we have modeled and evaluated operation of Bitcoin blockchain network in a setup where a mix of regular and compact blocks is used, and where nodes can leave the network for a random period. We have modeled high bandwidth operational mode with 3 TCP connections carrying compact blocks as well as low bandwidth operational mode with 3 and 6 TCP connections carrying compact blocks respectively. We have developed a detailed queuing model of

the transaction pool (mempool) at each node, including block and (optionally) mempool synchronization.

Our main findings can be summarized as follows. First, our results indicate that mempool synchronization is necessary to reduce the transaction deficit in such networks. Second, assuming compact blocks are transmitted over three TCP connections (in both high and low bandwidth modes), block and transaction delivery times drop by around 13% and 35%, respectively, while forking probability is reduced by around 22%. Finally, we have found that low bandwidth mode with increased number of TCP connections (six) carrying compact blocks outperforms high bandwidth mode by additional 15% in terms of reduction of block distribution time and forking probability. It also provides increased system security and resilience since blocks are validated at each node before forwarding, which prevents double spending attacks from dishonest miners.

## REFERENCES


[1] S. Ben Mariem, P. Casas, and B. Donnet. Vivisecting blockchain P2P networks: Unveiling the Bitcoin IP network. In *ACM CoNEXT Student Workshop*, 2018.

[2] Bitcoin improvement proposals. https://github.com/bitcoin/bips#readme, last accessed: July 26, 2020.

[3] P2P network guide. https://bitcoin.org/en/p2p-network-guide, last accessed: July 26, 2020.

[4] M. Corallo. BIP 152: compact block relay, 2016. https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki, last accessed July 26, 2020.

[5] S. Delgado-Segura, S. Bakshi, C. Pérez-Solà, J. Litton, A. Pachulski, A. Miller, and B. Bhattacharjee. TxProbe: Discovering Bitcoin's network topology using orphan transactions. *arXiv preprint arXiv:1812.00942*, 2018.

[6] A. E. Gencer, S. Basu, I. Eyal, R. Van Renesse, and E. G. Sirer. Decentralization in Bitcoin and Ethereum networks. *arXiv preprint arXiv:1801.03998*, 2018.

[7] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg. Eclipse attacks on Bitcoin's peer-to-peer network. In *USENIX Security Symposium*, pages 129–144, 2015.

[8] M. A. Imtiaz, D. Starobinski, A. Trachtenberg, and N. Younis. Churn in the bitcoin network: Characterization and impact. In *IEEE Int. Conf. on Blockchain and Cryptocurrency*, Seoul, South Korea, May 2019.

[9] L. J. Kleinrock. *Queuing Systems*, volume I: Theory. John Wiley and Sons, New York, 1972.

[10] J. F. Kurose and K. W. Ross. *Computer Networking: A Top-Down Approach Featuring The Internet*. Addison-Wesley Longman, Boston, MA, 6th edition, 2016.

[11] M. Lischke and B. Fabian. Analyzing the Bitcoin network: The first four years. *Future Internet*, 8(1):7, 2016.

[12] A. Miller, J. Litton, A. Pachulski, N. Gupta, D. Levin, N. Spring, and B. Bhattacharjee. Discovering Bitcoin's public topology and influential nodes. report, 2015.

[13] J. Mišić, V. B. Mišić, and X. Chang. On the benefits of compact blocks in Bitcoin. In *IEEE Int. Conf. on Communications ICC2020*, Dublin, Ireland, 2020.

[14] J. Mišić, V. B. Mišić, X. Chang, S. G. Motlagh, and M. Z. Ali. Modeling of Bitcoin's blockchain delivery network. *IEEE Transactions on Network Science and Engineering*, to appear, 2019.

[15] S. G. Motlagh, J. Mišić, and V. B. Mišić. Modeling of churn process in Bitcoin network. In *IEEE Int. Conference on Computing, Networking and Communications (ICNC 2020)*, Big Island, HI, 2020.

[16] K. Nayak, S. Kumar, A. Miller, and E. Shi. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 305–320, 2016.

[17] T. Neudecker, P. Andelfinger, and H. Hartenstein. Timing analysis for inferring the topology of the Bitcoin peer-to-peer network. In *Int. IEEE Conferences on Ubiquitous Intelligence Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*, pages 358–367, 2016. 10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0070.

[18] G. Pappalardo, G. Caldarelli, and T. Aste. The Bitcoin peers network. In *2nd Int. Workshop P2P Financial Systems*, London, UK, Sept. 2016.

[19] D. Ron and A. Shamir. Quantitative analysis of the full Bitcoin transaction graph. In *Int. Conf. Financial Cryptography and Data Security*, pages 6–24. Springer, 2013.

[20] H. Takagi. *Queueing Analysis*, volume 1: Vacation and Priority Systems. North-Holland, Amsterdam, The Netherlands, 1991.

[21] N. Younis, M. A. Imtiaz, D. Starobinski, and A. Trachtenberg. Improving bitcoin's resilience to churn. *CoRR*, abs/1803.06559, 2018.


**Jelena Mišić** (M'91, SM'08, F'18) is Professor of Computer Science at Ryerson University in Toronto, Ontario, Canada. She has published 4 books, over 125 papers in archival journals and close to 190 papers at international conferences in the areas of computer networks and security. She serves on editorial boards of *IEEE Transactions on Vehicular Technology*, *IEEE IoT Journal*, *IEEE Network*, *Computer Networks* and *Ad hoc Networks*. She is a Fellow of IEEE and Member of ACM.

**Vojislav B. Mišić** (M'92, SM'08) is Professor of Computer Science at Ryerson University in Toronto, Ontario, Canada. His research interests include performance evaluation of wireless networks and systems and software engineering. He serves on the editorial boards of *IEEE transactions on Cloud Computing*, *Ad hoc Networks*, *Peer-to-Peer Networks and Applications*, and *International Journal of Parallel, Emergent and Distributed Systems*. He is a Senior Member of IEEE and member of ACM.

**Xiaolin Chang** is Professor at the School of Computer and Information Technology, Beijing Jiaotong University. Her current research interests include edge/cloud computing, network security, security and privacy in machine learning.