RT-Blockchain: Achieving Time-Predictable Transactions

Seunghoon Lee, Sukmin Kang, Seungyeon Cho, Hyunwoo Koo, Sungjae Hwang, Jinkyu Lee*
Sungkyunkwan University (SKKU), Republic of Korea
{seunghoon.l, sukmin.kang, seungyeoncho, koowoo3, sungjaeh, jinkyu.lee}@skku.edu

Abstract*—Although blockchain technology is being increasingly utilized across various fields, the challenge of providing timing guarantees for transactions remains unmet, which is an obstacle in implementing blockchain solutions for time-sensitive applications such as high-frequency trading and real-time payments. In this paper, we propose the first solution to achieve a timing guarantee on blockchain. To this end, we raise and address two issues for timely transactions on a blockchain: (a) architectural support, and (b) real-time scheduling principles specialized for blockchain. For (a), we modify an existing blockchain network, offering an interface to preferentially select the transactions with the earliest deadlines. We then extend the blockchain network to provide the flexibility of the number of generated blocks at a single block time. Under such architectural supports, we achieve (b) with three steps. First, to resolve a discrepancy between a periodic request of a transaction-generating node and the corresponding arrival on a block-generating node, we translate the former into the latter, which eases the modeling of the transaction load imposed on the blockchain network. Second, we derive a schedulability condition of the modeled transaction load, which guarantees no missed deadline for all transactions under a work-conserving deadline-based scheduling policy. Last, we develop a lazy scheduling policy and its condition, which reduces the number of generated blocks without compromising the degree of timing guarantees for the work-conserving policy. By implementing RT-blockchain on top of an existing open-source blockchain project, we demonstrate the effectiveness of the proposed scheduling principles with architectural supports in not only ensuring timely transactions but also reducing the number of generating blocks.

I. Introduction

Blockchain technology, which started as cryptocurrencies is being widely used in various digital transactions starting from Non-fungible tokens (NFTs) that save digital content to smart contracts which are self-executing contracts wherein the contractual terms between a buyer and seller are encoded directly into lines of code [1]. However, blockchain is not being used in systems that need timing guarantees such as periodic status-report transactions (mobility as a service) [2], and time-sensitive economic transactions (payouts based on asset ownership or actions such as coupons) [3].

Time-predictability (in terms of timing guarantees) in blockchain systems has not been achieved since (a) the structure of blockchain does not innately offer a timing guarantee and (b) scheduling methodologies specialized for the blockchain architecture have never been proposed. Blockchain assures transactions by joining the block holding the transaction to the main chain. Transactions have to be selected and stored by the node making the block, and the selection mechanism, in most cases, runs on either first-come-first-serve or incentive-based algorithms. Thus, transactions can never be guaranteed to be distributed throughout the network at a particular time and have to wait until it is selected. Block time, the interval of blocks being generated, has been reduced, and block size has been increased from the original Bitcoin in many blockchain networks to enhance the transaction per second. Nevertheless, it is important to note that increase in transaction processing speed does not ensure precise transaction timing. Moreover, although there have been ongoing studies regarding the latency of transactions [4] [5] [6], there has been a lack of research on scheduling and ensuring the timely execution of deadline-oriented transactions within the blockchain system.

To implement a timing guarantee on blockchain, we address the challenges of (a) and (b), as follows.

A1. We create a blockchain network with an interface supporting transaction timing constraints and flexible block generation for accommodating more transactions within a single block time.
A2. We develop novel scheduling principles specialized for the blockchain network so as to guarantee the timely execution of periodic/sporadic transactions without wasting the blockchain network resources.

Regarding A1, our target network is constructed based on a well-known blockchain consensus mechanism called Proof of Stake (PoS). In PoS, users have the option to write high tipping fees to expedite their transactions; however, bidding fees can increase indefinitely, leaving users uncertain about the expected deployment time of their transactions. There are two main challenges to address (a) under the target network: first, there is no interface to express and utilize the deadline of each transaction, and second, increasing the rate of generating blocks itself cannot guarantee timely execution and incurs space inefficiency. Increasing block size or reducing block time can enhance transaction speed, but it comes at the cost of space efficiency. When fewer transactions are submitted to the blockchain than anticipated, idle blocks can accumulate, imposing a burden of storing the entire blockchain (including the idle blocks) on nodes that participate in the blockchain

*Jinkyu Lee is the corresponding author.

2576-3172/23/$31.00 ©2023 IEEE
DOI 10.1109/RTSS59052.2023.00018

network.

To address the first challenge for (a), we modify the target network. We allow nodes that send each transaction to put as input the expected deadline of the transaction. Then, it is feasible for the transactions with the earliest deadlines to be preferentially selected from the pool of transactions held by the block-generating node, while it depends on A2 how to utilize the feasibility for timing guarantees. For the second challenge for (a), we extend the target network to provide the flexibility of making zero to multiple blocks at a single block time. This method can prevent the generation of empty blocks and generate blocks as much as possible during block time depending on the number and size of transactions that are waiting in the transaction pool.

Meanwhile, addressing (b) via A2 is difficult due to two challenges. First, the timeline of the transaction request of a transaction-generating node is different from that of a block-generating node. Each transaction-generating node's inter-request time is not necessarily the same as one block time (or multiple thereof); even if it is the same, it cannot be synchronized with arrival on the block-generating node due to network delay. Second, each transaction cannot be split to be included in more than one block, which yields an arbitrarily small utilization of each block for priority-based approaches and therefore makes it difficult to schedule transactions without violating their timing constraints.

To address the two challenges for (b), we first translate a periodic/sporadic request of a transaction-generating node, into a periodic/sporadic arrival on a block-generating node, which eases the modeling of transaction load imposed on the blockchain network. We then derive a condition of the modeled transaction load; we guarantee no missed deadline for all transactions scheduled by a deadline-based scheduling policy, as long as the condition is satisfied. Finally, we develop a lazy deadline-based scheduling policy and its condition, which help reduce the number of generated blocks by efficiently skipping the block generation without incurring any missed deadline for all transactions.

We implemented RT-blockchain to an open-source blockchain, ran experiments accommodating various user-level periodic transaction task workloads, and proved that RT-blockchain meets timing constraints and reduces block generation by up to 40%, compared to work-conserving scheduling.

This paper makes the following contributions.

- We raise and address two issues to achieve timing guarantees for blockchain transactions, which is the first attempt.
- We modify the existing blockchain network enabling deadline-based transaction selection (in Section III) and extend it to provide the flexibility of the number of generated blocks at a single block time (in Section V).
- We develop novel scheduling principles that can guarantee the timely execution of periodic/sporadic transactions, which is the first achievement (in Section IV).
- We develop advanced scheduling methodologies that reduce the number of generated blocks without compromising timing guarantees of transactions (in Section VI).
- We implement the proposed network architecture and scheduling principles in a real blockchain, and evaluate their performance (in Section VII).

II. Understanding Blockchain for Real-Time

A. Blockchain Networks

Bitcoin [7], the pioneering blockchain technology, operates as a decentralized peer-to-peer network. Participating nodes competitively generate blocks to store transactions, receiving BTC (Bitcoin Cryptocurrency) as incentives. A consensus mechanism called Proof-of-Work (PoW) [8] is employed to maintain blockchain integrity. Nodes solve complex puzzles to propose new blocks, with the first successful node earning the right to create and append the next block. This resource-intensive process ensures network security and prevents transaction manipulation. BTC rewards incentivize participants, contributing to network maintenance and stability. However, the PoW consensus mechanism presents certain drawbacks. Its computational demands lead to substantial energy consumption, raising concerns about environmental impact. Additionally, as the network scales, transaction validation and consensus times increase, resulting in slower transaction processing.

In response to these challenges, alternative consensus mechanisms have emerged, including Proof-of-Stake (PoS) [9], which Ethereum [10] has transitioned to. PoS relies on validators who put a certain amount of cryptocurrency as collateral to create new blocks, eliminating the need for resource-intensive computations. This approach offers improved energy efficiency and the potential for faster transaction validation.

B. Applications that Need Timing Guarantees

Safety-critical systems, such as Mobility as a Service (MaaS) [2] platforms and time-sensitive trade applications, rely on timing guarantees. Periodic transactions play a vital role in advancing these applications. In MaaS, fulfilling timing guarantees ensures efficient transportation coordination, minimizing waiting times and enhancing platform reliability. Time-sensitive trade applications, like high-frequency trading [3], benefit from precise timing guarantees, enabling optimal strategies and improved profitability. Similarly, domains such as the monitoring of critical infrastructures [11] necessitate stringent timing guarantees to underpin real-time decision-making processes, thereby amplifying safety measures and mitigating operational disruptions. In parallel, collaborative coordination applications, like supply chain management [12] also benefit from timing guarantees, enabling synchronized activities and optimized resource utilization. Meeting timing guarantees of periodic transactions in safety-critical systems and time-sensitive trade applications unlock their maximum potential, enabling heightened efficiency, reliability, responsiveness, and collaborative capabilities to provide users with superior services and benefits.

III. RT-Blockchain System

In this section, we first explain the difficulties in achieving timing guarantees on our target blockchain architecture. Based on the architecture, we design RT-blockchain that offers an interface to preferentially select transactions by their deadlines. Also, we explain a timeline for processing transactions on a RT-blockchain.

A. *Timing Guarantee Issue in Blockchain Architecture*

Our design is based on a PoS consensus mechanism, known for energy efficiency and strong security. As a leading platform for smart contracts, Ethereum, while also using PoS, enables diverse decentralized applications and services. Users can include code in their transactions, fostering the adoption of smart contracts. These self-executing agreements, written as code on the blockchain, find applications in various fields like DeFi, supply chain management, voting systems, insurance claims, and intellectual property rights. Some adhere to predefined standards like ERC-20 for fungible tokens and ERC-721 for NFTs on the Ethereum network to ensure interoperability and interaction with other contracts and systems.

In Ethereum, users have the option to include a tip along with their transaction submission. This tip is provided to the block-generating node that includes the transaction in a block. Consequently, higher tipping prices result in faster confirmation and execution of the transaction. However, when multiple users seek to have their transactions executed within a single block time, the competitive increase in tipping prices leads to a backlog of transactions in the transaction pool.

In contrast, most private blockchains prioritize transactions based on a first-in-first-out (FIFO) order within their blocks. Private chains offer more control over transaction management since they operate within a single server and do not face synchronization issues among nodes. However, it is important to note that ensuring robust timing guarantees in safety-critical blockchain systems has yet to be thoroughly explored and implemented.

One of the challenges related to timing guarantees in blockchain systems is the competitive nature of transaction prioritization, as seen in public blockchains like Ethereum. This can result in delays and congestion when users vie for faster execution by offering higher tipping prices. In contrast, private blockchains may lack a robust mechanism for ensuring predictable timing guarantees due to their reliance on a single server and limited node synchronization.

As blockchain technology continues to evolve, it becomes crucial to address the challenges of timing guarantees in both public and private blockchains. This involves exploring new mechanisms or enhancements to existing protocols to optimize transaction prioritization, reduce congestion, and provide more reliable timing guarantees, particularly in safety-critical blockchain applications.

B. *RT-Blockchain: Offering Interface for Timely Transactions*

We design RT-blockchain, based on the Proof of Stake (PoS) consensus mechanism, which incorporates a modified interface

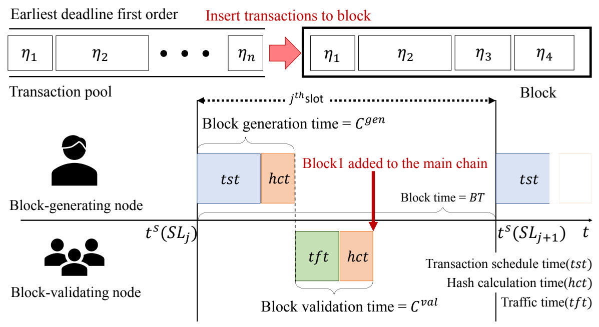

Earliest deadline first order
Insert transactions to block
η_1
η_2
η_n
η_1
η_2
η_3
η_4
Transaction pool
Block
j^{th}slot
Block generation time = C^{gen}
Block-generating node
tst
hct
Block1 added to the main chain
Block time = BT
tst
$t^s(SL_j)$
$t^s(SL_{j+1})$
t
Block-validating node
tft
hct
Block validation time = C^{val}
Transaction schedule time(tst)
Hash calculation time(hct)
Traffic time(tft)

Fig. 1. RT-blockchain architecture with a timeline for processing transactions

that enables the prioritization of transactions with the earliest deadlines. Figure 1 presents RT-blockchain architecture with a timeline of processing transactions. In the RT-blockchain system, a single block is generated and connected to the main blockchain at regular intervals (called *slots*), each of whose duration is the same and called block time (denoted by BT). The block time is predetermined and can vary depending on the specific blockchain protocol or cryptocurrency used. All transactions entering the system are stored in the transaction pool, and they are ordered based on their respective deadlines, with the earliest deadline transactions given priority.

Before the start of each slot, a designated node is selected to create the upcoming block. Once the slot begins, the selected block-generating node stops accepting additional transactions into its pool and starts selecting transactions to be included and then including them in the block. The block header, which contains information such as the previous block's hash, a timestamp, and transaction data, is used to calculate the block hash. A cryptographic hash function is applied to the block header data to calculate the block hash. The hash function employs complex mathematical operations on the input data to produce a fixed-size output, typically represented as a hexadecimal string of characters. The generated block hash ensures the integrity and sequential order of the blocks in the blockchain. It serves as a unique identifier for the block and is a crucial component for validation. We call this time that the block is made, block generation time.

Once the block generation process is completed, the block is distributed across the network. The verification process conducted by the validators is crucial for maintaining the reliability and security of the blockchain. By independently validating the transactions and verifying the block hash, the validators ensure that the transactions within the block are legitimate and that the block fits seamlessly into the blockchain's sequential order. By following this process for each block time, the RT-blockchain system creates a robust and trustworthy ledger of transactions. The interconnection of blocks through their hash values establishes a tamper-resistant chain, where any modification or tampering attempts can be easily identified and rejected. This process, starting from when the block is distributed to when it is added to the main chain, is defined as block validation time.

So far, we explained generating a block from the block-generating node's point of view; we now explain processing a transaction from the user's point of view using Figure 1.[1] When a user sends a transaction, it is transmitted to the blockchain network and reaches the transaction pool of the node responsible for generating blocks. The duration from the sender to the pool, representing the transaction's time within the network, is referred to as traffic time (whose upper bound is denoted by tft). The transaction then waits in the transaction pool until the next slot starts. Let $\texttt{SL}_j$ denote the j^{th} slot, and $t^s(\texttt{SL}_j)$ denote the time instant at which $\texttt{SL}_j$ starts. Once block generation begins at $t^s(\texttt{SL}_j)$, the node executes the transaction scheduling algorithm (to be explained in Section IV) and copies the transactions that have been scheduled into the block. The time required for a block to be filled with scheduled transactions is defined as the transaction schedule time (whose upper bound is denoted by tst). Once the block is filled and the transactions are finalized, the block generator computes the hash value of the block by utilizing the previous block's hash value and the hash value of the transactions. The time needed for this calculation is called hash calculation time (whose upper bound is denoted by hct). Then, an upper bound of block generation time by a block generating node, denoted by C^{gen}, is the sum of tst (time for scheduling transactions) and hct (time for computing the hash value) as shown in Figure 1.

The generated block is distributed throughout the network and reaches the local machines of each validator. We also calculate this distribution process to consume a time duration upper-bounded by tft. The validating nodes run hash calculations on the given block to check the validity for at most hct. Likewise, an upper bound of block validation time by a block validating node, denoted by C^{val}, is the sum of tft (network delay) and hct (time for computing the hash value) as shown in Figure 1. Therefore, a user's transaction request arriving in $(t^s(\texttt{SL}_{j-1}), t^s(\texttt{SL}_j)]$ (i.e., arriving later than the beginning of the $(j-1)^{th}$ slot but no later than the beginning of the j^{th} slot) on a block-generating node can finish its execution no later than $t^s(\texttt{SL}_j) + C^{gen} + C^{val}$, if the transaction is included in the block at the j^{th} slot.

Note that a transaction cannot be split and stored due to the fundamental design of traditional blockchains. The integrity of the blockchain depends on the cohesive nature of each block, where transactions are stored as complete entities, and the block's hash is calculated based on the contents of those transactions. Splitting a transaction would compromise the integrity of the blockchain, as it would disrupt the interdependencies among transactions and their cohesive representation within blocks. Furthermore, the complexity and manageability of validating transactions would significantly increase if they were split and stored separately. The fragmented transactions would need to be linked across different blocks, introducing complexities in verifying their validity and maintaining consistency within the blockchain.

[1]In this paper, we will use the term of a transaction-generating node and that of a user interchangeably.

IV. Timely Transactions on RT-Blockchain

In this section, we achieve timing guarantees of a set of periodic/sporadic user-level (i.e., transaction-generating-node-level) blockchain transaction tasks on a RT-blockchain. To this end, we first establish the goal and raise challenges for timing guarantees of a user-level task set. We then define a notion of a slot-level (i.e., block-generating-node-level) blockchain transaction task, and translate a user-level task to the corresponding slot-level one without compromising the user-level timing constraint. Finally, we achieve timing guarantees for a translated slot-level task set by developing a schedulability condition under a deadline-based prioritization policy, which is the first achievement for a blockchain.

A. Blockchain Transactions in Time: Goal and Challenges

According to the timing requirements of transactions presented in Section II-B, we define a user-level blockchain transaction task subject to a timing constraint as follows.

Definition 1: We define a periodic/sporadic user-level (i.e., transaction-generating-node-level) blockchain transaction task $\eta_i \in \eta$, specified by (T_i, D_i, S_i), where T_i is the inter-arrival time (or called period), D_i is the relative deadline, and S_i is the maximum transaction size. A user-level blockchain transaction task $\eta_i(T_i, D_i, S_i)$ generates a series of transaction jobs, the x^{th} of which is denoted by $\eta_{i,x}$. The release times of η_i's two consecutive jobs $\eta_{i,x}$ and $\eta_{i,x+1}$ are separated by at least T_i. Once a job $\eta_{i,x}$ is released at t, its transaction whose size is at most S_i should be completed no later than $t + D_i$; otherwise, we express $\eta_{i,x}$ misses its deadline.

Note that the relative deadline D_i could be smaller than, equal to, or larger than T_i, depending on the corresponding user's requirement.

Recall that a blockchain transaction is regarded as completed when each block-validating node finishes the validation process as presented in Section III-B. Therefore, a user's transaction request arriving in $(t^s(\texttt{SL}_{j-1}), t^s(\texttt{SL}_j)]$ on a block-generating node can finish its execution no later than $t^s(\texttt{SL}_j) + C^{gen} + C^{val}$ shown in Figure 1, if the transaction is included in the block at the j^{th} slot.

We define the schedulability of a periodic/sporadic user-level blockchain transaction task set η as follows.

Definition 2: A periodic/sporadic user-level blockchain transaction task set η is said to be *schedulable*, if there is no single job deadline miss for every legal job series generated by η.

Using the definitions, we state our goal as follows.

Problem Statement: We develop a deadline-based scheduling algorithm *and* a schedulability condition for η, which is scheduled by the scheduling algorithm on a RT-blockchain system, such that

G1. η is schedulable if η satisfies the schedulability condition (achieving *timing guarantees*).

Although the parameters for $\eta_i \in \eta$ look similar to those of the traditional real-time periodic/sporadic task $\tau_i \in \tau$ (specified by T_i, D_i and the worst-case execution time), achieving G1 in the problem statement is totally different from the case for the traditional real-time periodic/sporadic task set τ executed on a computing platform due to the following challenges.

C1. The timeline of each user's (i.e., transaction-generating node's) transaction request is different from that of block generation. Each user's inter-request time (the time period between η_i's two consecutive job requests) is not necessarily the same as one block time (or multiple thereof); even if it is the same, the transaction request inter-arrival time (the time period between η_i's two consecutive job arriving in a block-generating node) cannot be synchronized with each user's inter-request time due to network delay.

C2. Each transaction cannot be split to be included in more than one block, which may yield an arbitrarily small utilization of each block under priority-based approaches. For example, due to a higher-priority transaction job whose size is 10% of the block size, a lower-priority one whose size is 95% of the block size cannot be included in the same block, yielding 10% utilization of the block in spite of the total workload of 105%.

To address C1 first, we need to interpret the release time and the absolute deadline of each user's transaction job, from the block-generating node's point of view.

B. User-level to Slot-level Translation

From the block-generating node's point of view, we define a set of sporadic slot-level blockchain transaction task subject to a timing constraint as follows.

Definition 3: We define a sporadic slot-level blockchain transaction task $\eta_i^{\mathsf{S}} \in \eta^{\mathsf{S}}$, specified by $(T_i^{\mathsf{S}}, D_i^{\mathsf{S}}, S_i^{\mathsf{S}}, N_i^{\mathsf{S}})$, where T_i^{S} is the inter-arrival slots, D_i^{S} is the slot-unit relative deadline, S_i^{S} is the maximum transaction size, and N_i^{S} is the number of transactions of each job of η_i^{S}. A slot-level blockchain transaction task $\eta_i^{\mathsf{S}}(T_i^{\mathsf{S}}, D_i^{\mathsf{S}}, S_i^{\mathsf{S}})$ generates a series of transaction jobs, the x^{th} of which is denoted by $\eta_{i,x}^{\mathsf{S}}$. The ready-to-be-scheduled time instant for $\eta_{i,x}^{\mathsf{S}}$ is $t^s(\mathtt{SL}_j)$, (i.e., the beginning of the j^{th} slot), if $\eta_{i,x}^{\mathsf{S}}$ that arrives in a block-generating node within the interval later than $t^s(\mathtt{SL}_{j-1})$ but no later than $t^s(\mathtt{SL}_j)$ i.e., the interval of $(t^s(\mathtt{SL}_{j-1}), t^s(\mathtt{SL}_j)]$. The ready-to-be-scheduled time instants of two consecutive jobs $\eta_{i,x}^{\mathsf{S}}$ and $\eta_{i,x+1}^{\mathsf{S}}$ are separated by at least T_i^{S} slot units. Once a job $\eta_{i,x}^{\mathsf{S}}$ is ready-to-be-scheduled at $t^s(\mathtt{SL}_j)$, there are N_i^{S} transactions whose size is at most S_i^{S}. Each of them should be included in any block of $\mathtt{SL}_j$, $\mathtt{SL}_{j+1}$, ..., or $\mathtt{SL}_{j+D_i^{\mathsf{S}}-1}$,[2] without being split to be included in more than one block; otherwise, we express $\eta_{i,x}^{\mathsf{S}}$ misses its deadline.

Note that the slot-unit relative deadline D_i^{S} could be smaller than, equal to, or larger than T_i^{S}.

[2]In this case, $\mathtt{SL}_{j+D_i^{\mathsf{S}}-1}$ is the slot-unit absolute deadline.

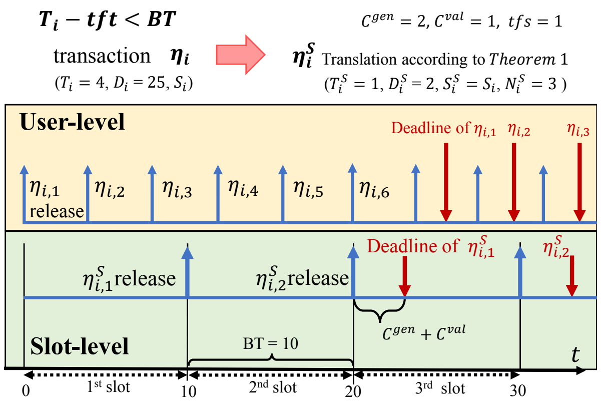

$T_i - tft < BT$
$C^{gen} = 2, C^{val} = 1,\ tfs = 1$
transaction η_i
$(T_i = 4, D_i = 25, S_i)$
η_i^S Translation according to $Theorem$ 1
$(T_i^S = 1, D_i^S = 2, S_i^S = S_i, N_i^S = 3)$
User-level
Deadline of $\eta_{i,1}$ $\eta_{i,2}$ $\eta_{i,3}$
$\eta_{i,1}$ release
$\eta_{i,2}$
$\eta_{i,3}$
$\eta_{i,4}$
$\eta_{i,5}$
$\eta_{i,6}$
Deadline of $\eta_{i,1}^S$ $\eta_{i,2}^S$
$\eta_{i,1}^S$release
$\eta_{i,2}^S$release
$C^{gen} + C^{val}$
Slot-level
BT = 10
t
0
1st slot
10
2nd slot
20
3rd slot
30

Fig. 2. User-level to slot-level translation

For example, when a user sends a transaction $(T_i = 4, D_i = 25, S_i)$, it arrives at the block-generating node at most after tft. As illustrated in Figure 2, where the block time BS is 10, in most cases the release of transactions on the user-level will not be synchronous with the release of block generation. Thus we need to translate (T_i, D_i, S_i) to $(T_i^{\mathsf{S}}, D_i^{\mathsf{S}}, S_i^{\mathsf{S}}, N_i^{\mathsf{S}})$ for the ease of scheduling multiple transactions to blocks. For this translation, we first need to consider the factors that influence the absolute deadline of transaction η_i. Consider the time when the user releases a transaction at $t = 0$, the actual time the block-generating node will receive this transaction on their pool will be no later than $t = tft$. Thus, $\eta_{i,1}$ in Figure 2, although expected for release at the user-level at $t = 0$, due to the traffic time (tft), will be assumed to ready-to-be-scheduled in the beginning of next slot $t = t^s(\mathsf{SL}_2) = 10$; this is because it does not arrive at the first slot until the beginning of the slot $t = t^s(\mathsf{SL}_1) = 0$. More evidently, transactions $\eta_{i,2}$ and $\eta_{i,3}$ which were expected to release after $t = 0$ but before $t = 10 - tft$ must be released before $t = 10$. Therefore, we can translate the transactions $\eta_{i,1}$, $\eta_{i,2}$, and $\eta_{i,3}$ as $\eta_{i,1}^{\mathsf{S}}$ in the slot-level expression and increase the number of transactions by $N_{i,1}^{\mathsf{S}} = 3$. The earliest-deadline transaction $\eta_{i,1}$ among the three transactions $\eta_{i,1}$, $\eta_{i,2}$, and $\eta_{i,3}$ should be included in the block of the second or third slot; since its deadline is between $[20 + C^{gen} + C^{val}, 30 + C^{gen} + C^{val})$, it misses its deadline if it is included in the block of the fourth or later slot. Since $\eta_{i,1}$ needs to be included in the block of one of the two slots (i.e., the second or third slot), its corresponding $D_{i,1}^{\mathsf{S}} = 2$.

We then define the schedulability of a set of periodic/sporadic slot-level blockchain transaction tasks η^{S}, similar to that of η as follows.

Definition 4: A periodic/sporadic slot-level blockchain transaction task set η^{S} is said to be *schedulable*, if there is no single job deadline miss for every legal job series generated by η^{S}.

Then, we can translate η to η^{S}, without compromising timing constraints of every $\eta_i \in \eta$. We consider two cases to derive T_i^{S} and N_i^{S}. If $(T_i - tft) \geq BT$ holds, we can guarantee that the period between η_i's two consecutive jobs' arrivals in a block-generating node is not larger than $(T_i - tft)$ as long as the network delay is upper-bounded by tft. Therefore, the interval-arrival slot units of η_i is upper-bounded by Eq. (3), and there is at most one job of η_i arriving in a single slot. Conversely, if $(T_i - tft) < BT$ holds, the number of transaction jobs of η_i arriving in a block-generating node in a single slot is upper-bounded by Eq. (4). Using this reasoning, the following theorem translates η to η^{S} without compromising the schedulability of η.

Theorem 1: The schedulability of a periodic/sporadic user-level blockchain transaction task set η is schedulable on a RT-blockchain system, if its corresponding periodic/sporadic slot-level blockchain transaction task set η^{S} is schedulable, where parameters of each $\eta_i^{\mathsf{S}} \in \eta^{\mathsf{S}}$ is set as follows.

$$D_i^{\mathsf{S}} = \Big\lfloor \frac{D_i - tft - C^{gen} - C^{val}}{BT} \Big\rfloor, \quad (1)$$

$$S_i^{\mathsf{S}} = S_i, \quad (2)$$

$$\text{If } T_i - tft \geq BT, \quad T_i^{\mathsf{S}} = \Big\lfloor \frac{T_i - tft}{BT} \Big\rfloor, \text{ and } N_i^{\mathsf{S}} = 1, \quad (3)$$

$$\text{If } T_i - tft < BT, \quad T_i^{\mathsf{S}} = 1, \text{ and } N_i^{\mathsf{S}} = \Big\lceil \frac{BT + tft}{T_i} \Big\rceil. \quad (4)$$

Note that if $D_i^{\mathsf{S}} \leq 0$ holds for any $\eta_i \in \eta$, η^{S} is not schedulable under any scheduling.

Proof: In any case, the transaction size does not change; therefore, $S_i^{\mathsf{S}} = S_i$ holds. To derive D_i^{S}, suppose a job of η_i experiences the maximum network delay (upper-bounded by tft) and arrives in a block-generating node right after $t^s(\mathtt{SL}_j)$ (the beginning of the j^{th} slot), which is the worst-case scenario as it is ready-to-be-scheduled at $t^s(\mathtt{SL}_{j+1})$. In this scenario, if its absolute deadline is no earlier than $t^s(\mathtt{SL}_{j+1})+C^{gen}+C^{val}$ (which is equivalent to $D_i - BT - tft - C^{gen} - C^{val} \geq 0$), it does not miss its deadline in case that the transaction is included in the block of the $(j+1)^{th}$ slot. Therefore, D_i^{S} is not smaller than $\lfloor \frac{D_i - BT - tft - C^{gen} - C^{val}}{BT} \rfloor + 1$, which is equivalent to Eq. (1).

To derive T_i^{S} and N_i^{S}, we consider two cases. First, consider the case of $(T_i - tft) \geq BT$. The scenario, where the inter-arrival time is minimized, occurs when $\eta_{i,x}$ experiences the largest network delay and $\eta_{i,x+1}$ experiences no network delay, and the inter-arrival time in the scenario is $(T_i - tft)$ by considering the network delay is no larger than tft. Therefore, Eq. (3) holds.

Second, consider the case of $(T_i - tft) < BT$. The scenario, where the number of jobs of η_i to arrive in a single slot is maximized, occurs when the first and last jobs of η_i that arrive in a single slot experience the largest and no network delay, respectively, which yields Eq. (4), by considering the network delay is no larger than tft.

Since we derive the smallest possible T_i^{S} and D_i^{S} and the largest possible N_i^{S}, the theorem holds. ∎

C. Timing Guarantees for RT-Blockchain

Thanks to Theorem 1, it is sufficient to develop a scheduling algorithm and schedulability analysis for the corresponding η^{S} (instead of η) to achieve G1 for η in Section IV-A as follows.

Problem Statement: We develop a deadline-based scheduling algorithm *and* a schedulability condition for η^{S} scheduled by the scheduling algorithm on a RT-blockchain system, such that G1$^{\mathsf{S}}$ is satisfied.

G1$^{\mathsf{S}}$. η^{S} is schedulable if η^{S} satisfies the schedulability condition (achieving *timing guarantees*).

As to the scheduling algorithm for η^{S}, we apply EDF$^{\mathtt{WC}}$ (Work-Conserving Earliest Deadline First) for a RT-blockchain system as follows.

*Algorithm of EDF$^{\mathtt{WC}}$ for η^{S} with a single block*: At every beginning of each slot (e.g., $t^s(\mathtt{SL}_j)$ for the j^{th} slot), sort all transactions in the waiting queue by their absolute slot-unit deadlines. Then, assign each sorted transaction to the block one by one, until the highest-priority unassigned transaction in the waiting queue cannot be assigned due to the remaining portion of the block is smaller than the size of the highest-priority unassigned transaction.

To develop a schedulability test for η^{S} under EDF$^{\mathtt{WC}}$, we define a demand bound function and load for η_i^{S}, modified by those for the traditional real-time periodic/sporadic task [13].

Definition 5: Let $\mathtt{DBF}(\eta_i^{\mathsf{S}}, q)$ denote the demand bound function for a slot-level blockchain transaction task $\eta_i^{\mathsf{S}} \in \eta^{\mathsf{S}}$ during q consecutive slots (where $q = 1, 2, 3, ...$), defined by

$$\mathtt{DBF}(\eta_i^{\mathsf{S}}, q) = \max\left(0, \left(\Big\lfloor \frac{q - D_i^{\mathsf{S}}}{T_i^{\mathsf{S}}} \Big\rfloor + 1\right) \cdot \frac{S_i^{\mathsf{S}} \cdot N_i^{\mathsf{S}}}{BS}\right). \quad (5)$$

Definition 6: Let $\mathtt{LOAD}(\eta^{\mathsf{S}})$ denote the maximum of the sum of all $\mathtt{DBF}(\eta_i^{\mathsf{S}}, q)$ for $\eta_i^{\mathsf{S}} \in \eta_i$ divided by the number of consecutive slots q, which is defined by

$$\mathtt{LOAD}(\eta^{\mathsf{S}}) = \max_{q=1,2,3,...} \frac{\sum_{\eta_i^{\mathsf{S}} \in \eta^{\mathsf{S}}} \mathtt{DBF}(\eta_i^{\mathsf{S}}, q)}{q}. \quad (6)$$

Assuming the system starts the first slot among the consecutive q slots, the meaning of $\mathtt{DBF}(\eta_i^{\mathsf{S}}, q)$ is the total size of transaction jobs of η_i^{S} normalized by the block size that should be handled during q slots, and that of $\mathtt{LOAD}(\eta^{\mathsf{S}})$ is the maximum of the total size of transaction jobs of every $\eta_i^{\mathsf{S}} \in \eta_i$ normalized by the number of consecutive slots q.

If we consider the analogy between $\mathtt{DBF}(\eta_i^{\mathsf{S}}, q)$ and that for the traditional real-time periodic/sporadic task, one may think that $\mathtt{LOAD}(\eta^{\mathsf{S}}) \leq 1.0$ could be a schedulability condition for η^{S}; however, this is wrong due to C2 presented in Section IV-A. By addressing C2, we develop a schedulability condition for η^{S}, as follows.

Theorem 2: A set of periodic/sporadic slot-level blockchain transaction tasks η^{S} is schedulable by EDF$^{\mathtt{WC}}$ on a RT-blockchain system, if the following condition holds.

$$\mathtt{LOAD}(\eta^{\mathsf{S}}) \leq 1.0 - \max_{\eta_i \in \eta^{\mathsf{S}}} \frac{S_i^{\mathsf{S}}}{BS} \quad (7)$$

Proof: First, we prove that in any slot, if the sum of the size of transactions in the waiting queue is larger than $BS \cdot \left(1.0 - \max_{\eta_i \in \eta^{\mathsf{S}}} \frac{S_i^{\mathsf{S}}}{BS}\right)$, the block can include a set of transactions whose size summation is at least $BS \cdot \left(1.0 - \max_{\eta_i \in \eta^{\mathsf{S}}} \frac{S_i^{\mathsf{S}}}{BS}\right)$. Suppose that the statement is wrong. Then, there exists a job that cannot be included in the block. Since the block has at least $BS \cdot \max_{\eta_i \in \eta^{\mathsf{S}}} \frac{S_i^{\mathsf{S}}}{BS}$ amount remaining space, this violates the policy of EDF$^{\mathtt{WC}}$.

Second, using the property, the remaining proof is similar to demand-bound-function-based EDF schedulability analysis for the traditional real-time periodic/sporadic tasks [13] as follows. Suppose that there exists a deadline miss of a transaction of $\eta_{i,x}$ at the n^{th} slot. Let n_0 $(\leq n)$ denote the largest index of the slot, such that (i) $BS \cdot \left(1.0 - \max_{\eta_i \in \eta^{\mathsf{S}}} \frac{S_i^{\mathsf{S}}}{BS}\right)$ is strictly larger than the sum of the size of transactions whose slot-unit absolute deadline is no later than that of the transaction of $\eta_{i,x}$ in the block of $(n_0 - 1)^{th}$ slot while (ii) $BS \cdot \left(1.0 - \max_{\eta_i \in \eta^{\mathsf{S}}} \frac{S_i^{\mathsf{S}}}{BS}\right)$ is no larger than that in the block of n_0^{th} slot. The first property we proved implies that there is no job whose release slot index is $n_0 - 1$ or smaller, but is not included in any block until the beginning of the n_0^{th} slot. Therefore, $BS \cdot \sum_{\eta_i^{\mathsf{S}} \in \eta^{\mathsf{S}}} \mathtt{DBF}(\eta_i^{\mathsf{S}}, n - n_0 + 1)$ is the largest possible sum of the size of transactions whose slot-unit absolute deadline is no later than that of $\eta_{i,x}$. Also, by the definition of n_0 and n, $(n - n_0 + 1) \cdot BS \cdot \left(1.0 - \max_{\eta_i \in \eta^{\mathsf{S}}} \frac{S_i^{\mathsf{S}}}{BS}\right)$ is a lower-bound of the sum of the size of transactions whose slot-unit absolute deadline is no later than that of $\eta_{i,x}$ included in the block of either n_0^{th}, $(n_0 + 1)^{th}$, ..., or n^{th} slot. This contradicts Eq. (7). ■

V. Multi-Block RT-Blockchain System Design

Based on raising scalability issues in Blockchain, this section designs multi-block RT-blockchain that offers the flexibility of the number of generated blocks at a single block time. The flexibility offers the opportunity to accommodate more transactions subject to timing constraints without wasting the blockchain network resources, to be addressed in Section VI.

A. Scalability Issue in Blockchain

In most blockchain systems, such as Ethereum, a single fixed-sized block is generated at regular intervals, regardless of the number of transactions waiting in the transaction pool. This approach can lead to the creation of empty blocks when no transactions are submitted during a block time. Empty blocks are problematic as they consume computational resources and storage space without contributing to transaction processing. A notable example illustrating the impact of transaction congestion is the case of Cryptokitties, an NFT [14] application on the Ethereum network. The surge in Cryptokitties transactions resulted in significantly lower transaction speeds and a substantial number of pending transactions in the pool, reaching up to 20,000 to 25,000. This highlighted the blockchain community's emphasis on enhancing transaction speed, measured in transactions per second, to eliminate delays.

To address scalability concerns and enhance transaction throughput, blockchain systems employ various solutions [15]. Increasing the block size allows more transactions to be included in each block, accommodating a higher volume of transactions. Similarly, reducing the block time enables more frequent block creation within a given time period. These approaches aim to improve transaction speed and alleviate congestion. However, they also face trade-offs when the expected transaction volume is not met, leading to the generation of empty blocks. To further enhance scalability, alternative ideas such as sharding [16] and off-chain transactions are being studied. Sharding involves dividing the blockchain network into smaller partitions called shards, enabling parallel processing of transactions and increasing overall throughput and capacity. Off-chain transactions, facilitated by techniques like payment channels (e.g., the Lightning Network [17]) and state channels, move certain transactions off the main blockchain, reducing on-chain load and congestion. Layer 2 solutions [18] provide additional layers on top of the main blockchain, enabling off-chain transaction processing while leveraging the security of the underlying blockchain. Sidechains [19], Plasma [20], and Rollups [21] are examples of layer 2 protocols that can handle a significant number of transactions off-chain, periodically settling them on the main chain to improve scalability while maintaining security. However, it is important to note that these methods do not directly address the resource-wasting problem by empty block caused by the lack of anticipated transaction volume. Further research and development are needed to tackle this specific issue and optimize blockchain systems for efficient resource utilization and transaction processing.

B. Design of Multi-Block RT-Blockchain

On top of RT-Blockchain explained in Section III-B, we design an architecture that generates multiple blocks per block time to enhance the scalability of the blockchain system while preventing the generation of empty blocks. To distinguish, we call this a *multi-block* RT-blockchain, while the previous one is called a *single-block* RT-blockchain. Figure 3 presents architecture and timeline of multi-block RT-blockchain. Every slot, when the block-generating node checks the transaction pool, it decides how many blocks it should generate on that single slot within a predefined threshold of the maximum number of generating blocks in each slot (denoted by m). When a block is made, it is deployed directly to the network for validation. Due to the flexibility of a number of generated blocks, no empty blocks are made, and the trade-off between transaction speed and waste of computational space is alleviated. Our architecture addresses scalability challenges in the blockchain system by generating multiple blocks per block time. Unlike traditional blockchain systems, our architecture allows the block-generating node to determine the number of blocks to generate during each block time based on the transactions in the pool. This approach ensures that there are no empty blocks generated, eliminating the inefficiency associated with wasted computational resources and storage space.

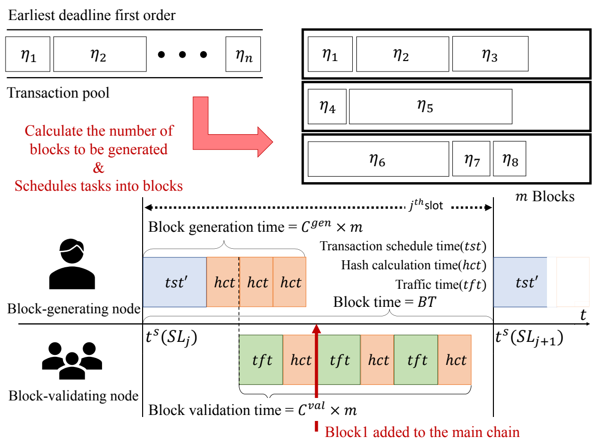

Earliest deadline first order
η_1 η_2 η_n
Transaction pool
Calculate the number of blocks to be generated & Schedules tasks into blocks
η_1 η_2 η_3 η_4 η_5 η_6 η_7 η_8
m Blocks
j^{th}slot
Block generation time = $C^{gen} \times m$
Transaction schedule time(tst)
Hash calculation time(hct)
Traffic time(tft)
Block time = BT
tst' hct hct hct tst'
Block-generating node
t
$t^s(SL_j)$
$t^s(SL_{j+1})$
tft hct tft hct tft hct
Block-validating node
Block validation time = $C^{val} \times m$
Block1 added to the main chain

Fig. 3. Architecture and timeline of multi-block RT-blockchain

In the multi-block RT-blockchain architecture, when each block is created at t (e.g., $t^s(\mathtt{SL}_j)+tst'+hct$ in Figure 3 for the first block of the j^{th} slot), it immediately starts propagating to the block-validating node without waiting for creating the next blocks in the same slot. Thereby, the validation of the first block generated in the j^{th} slot is finished no later than $t^s(\mathtt{SL}_j)+tst'+hct+tft+hct)$ shown in Figure 3 as "Block1 added to the main chain". This mechanism ensures a seamless and efficient transaction processing flow. By dynamically adjusting the number of blocks generated per block time, we strike a balance between transaction speed and resource utilization, optimizing the system for enhanced scalability. By preventing the generation of empty blocks, our architecture maximizes the usage of each block, effectively utilizing computational resources and storage space. This approach significantly improves the efficiency of the blockchain system and minimizes any potential waste in the blockchain system that occurs due to idle blocks.

The node generating the block first calculates how many blocks will be needed to fit the transaction in the pool. Then the node stores transactions sequentially in each block while keeping the bound that will be explained in Section 6. Considering the transaction schedule time for m blocks (whose upper-bound is denoted by tst') can be larger than that for a single-block RT-blockchain for one block (whose upper-bounded is denoted by tst), we assume that $tst' \leq m \cdot tst$, implying block generation time for a m-block RT-blockchain is upper-bounded by $tst' + m \cdot hct \leq m \cdot (tst + hct) = m \cdot C^{gen}$. Also, block validation time is upper-bounded by $m \cdot C^{val} = m \cdot (tft+hct)$. Therefore, any transaction included in one of the m blocks at the j^{th} slot is finished no later than $t^s(\mathtt{SL}_j) + m \cdot C^{gen} + m \cdot C^{val}$.

VI. Achieving Timing Guarantees and Resource Efficiency on Multi-Block RT-Blockchain

Extending the results of timing guarantees on a single-block RT-blockchain in Section IV, we first achieve timing guarantees on a multi-block RT-blockchain. Then, we address the issue of resource efficiency (i.e., reducing the number of generated blocks) without compromising timing guarantees. To this end, we develop a lazy scheduling algorithm that safely delays transactions to some later blocks, exploit it to fully utilize the blocks inevitably generated to avoid a deadline miss, and develop a schedulability condition under the lazy algorithm, which is also the first attempt that address both timing guarantees and resource efficiency for a blockchain.

A. *Timing Guarantees for Multi-Block RT-Blockchain*

To provide timing guarantees for a user-level blockchain transaction task set η executed on a multi-block blockchain system, we also translate η into η^{S} by applying a similar approach to Theorem 1 as follows; the only difference is D_i^{S}.

Theorem 3: The schedulability of a periodic/sporadic user-level block-chain transaction task set η^{S} is schedulable on a m-block blockchain system, if its corresponding periodic/sporadic slot-level block-chain transaction task set η^{S} is schedulable, where D_i^{S} of each η_i^{S} is set to Eq. (8) and other parameters are set to Eqs. (2), (3) and (4) in Theorem 1.

$$D_i^{\mathsf{S}} = \left\lfloor \frac{D_i - tft - m \cdot C^{gen} - m \cdot C^{val})}{BT} \right\rfloor \tag{8}$$

Note that if $D_i^{\mathsf{S}} \leq 0$ holds for any $\eta_i \in \eta$, η^{S} is not schedulable under any scheduling.

Proof: The proof is the same as that of Theorem 1, except considering a transaction job included in the m^{th} block in the j^{th} slot is completed no later than $t^s(\mathtt{SL}_j)+m\cdot C^{gen}+m\cdot C^{val}$ instead of $t^s(\mathtt{SL}_j) + C^{gen} + C^{val}$, which changes D_i^{S}. ■

Once η is translated to η^{S} by Theorem 3, our goal is to develop a deadline-based scheduling algorithm and a schedulability condition for η^{S} scheduled by the scheduling algorithm on a multi-block blockchain system, such that G1$^{\mathsf{S}}$ in Section IV-C holds.

When it comes to the scheduling algorithm, we also apply EDF$^{\mathtt{WC}}$, generalized for m blocks (instead of a single block) to be supplied.

Algorithm of EDF$^{\mathtt{WC}}$ for η^{S} with m blocks: At every beginning of each slot (i.e., $t^s(\mathtt{SL}_j)$), sort all transactions in the waiting queue by their absolute slot-unit deadlines. Then, assign each sorted transaction to one of the m blocks one by one using the first-fit policy, until the highest-priority unassigned transaction in the waiting queue cannot be assigned to any of the m blocks due to the remaining portion of each of the m blocks is smaller than the size of the highest-priority unassigned transaction.

Then, the schedulability analysis for the single-block RT-blockchain in Theorem 2 is generalized for the multi-block RT-blockchain as follows.

Theorem 4: A set of periodic/sporadic slot-level blockchain transaction tasks η^{S} is schedulable by EDF$^{\mathtt{WC}}$ on a m-block blockchain system, if the following condition holds.

$$\mathtt{LOAD}(\eta^{\mathsf{S}}) \leq \mathtt{LOAD}^*(\eta^{\mathsf{S}}), \tag{9}$$

$$\text{where } \texttt{LOAD}^*(\eta^{\mathsf{S}}) = m \cdot \left(1.0 - \max_{\eta_i \in \eta^{\mathsf{S}}} \frac{S_i^{\mathsf{S}}}{BS}\right). \tag{10}$$

Proof: The proof holds by that of Theorem 2 by substituting $\left(1.0 - \max_{\eta_i \in \eta^{\mathsf{S}}} \frac{S_i^{\mathsf{S}}}{BS}\right)$ by $m \cdot \left(1.0 - \max_{\eta_i \in \eta^{\mathsf{S}}} \frac{S_i^{\mathsf{S}}}{BS}\right)$. ■

However, the load upper-bound for Theorem 4 is pessimistic when $\left(1.0 - \max_{\eta_i \in \eta^{\mathsf{S}}} \frac{S_i^{\mathsf{S}}}{BS}\right)$ is small. We can improve the upper-bound as follows.

Theorem 5: A set of periodic/sporadic slot-level blockchain transaction tasks η^{S} is schedulable by EDF$^{\texttt{WC}}$ on a m-block blockchain system, if the following condition holds.

$$\texttt{LOAD}(\eta^{\mathsf{S}}) \le \texttt{LOAD}^{**}(\eta^{\mathsf{S}}), \tag{11}$$

$$\begin{aligned}\text{where } \texttt{LOAD}^{**}(\eta^{\mathsf{S}}) =& \max\left(\frac{1}{2}, 1.0 - \max_{\eta_i \in \eta^{\mathsf{S}}} \frac{S_i^{\mathsf{S}}}{BS}\right) \cdot (m-1) \\ &+ \left(1.0 - \max_{\eta_i \in \eta^{\mathsf{S}}} \frac{S_i^{\mathsf{S}}}{BS}\right).\end{aligned} \tag{12}$$

Proof: Since Theorem 5 is equivalent to Theorem 4 in case of $\left(1.0 - \max_{\eta_i \in \eta^{\mathsf{S}}} \frac{S_i^{\mathsf{S}}}{BS}\right) \ge 0.5$, we prove the case for $\left(1.0 - \max_{\eta_i \in \eta^{\mathsf{S}}} \frac{S_i^{\mathsf{S}}}{BS}\right) < 0.5$.

Applying the same proof technique as Theorem 2, it suffices to prove the following: in any slot, if the sum of the size of transactions in the waiting queue is larger than $BS \cdot \texttt{LOAD}^{**}(\eta^{\mathsf{S}})$, the block can include a set of transactions whose size summation is at least $BS \cdot \texttt{LOAD}^{**}(\eta^{\mathsf{S}})$ without violating the priority of transactions (determined by their deadlines).

Suppose that there are at least two blocks, each of which is occupied no larger than 50% (i.e., less than $\frac{1}{2} \cdot BS$ amount of the total size of transactions included). Then, it immediately contradicts the first-fit policy in EDF$^{\texttt{WC}}$; in other words, all the transactions in the two blocks should be included in one of the blocks. Therefore, there is at most one block that is occupied no larger than 50%. If there is no such block, the sum of the size of transactions in the m blocks is no smaller than $BS \cdot \frac{m}{2}$, which is larger than $BS \cdot \texttt{LOAD}^{**}(\eta^{\mathsf{S}})$. Therefore, the remaining proof is to consider the case of the existence of only one block that is occupied no larger than 50%. If the sum of the size of transactions in the block is small than $\left(1.0 - \max_{\eta_i \in \eta^{\mathsf{S}}} \frac{S_i^{\mathsf{S}}}{BS}\right)$, it violates the policy of EDF$^{\texttt{WC}}$ (by the same proof as the first part of Theorem 2). Otherwise, the sum of the size of transactions in the m blocks is no smaller than $BS \cdot \texttt{LOAD}^{**}(\eta^{\mathsf{S}})$. This proves the theorem. ■

Since no scheduler can make η^{S} schedulable if $\texttt{LOAD}(\eta^{\mathsf{S}}) > m$, the resource augmentation bound for EDF$^{\texttt{WC}}$ on a multi-block RT-blockchain is

$$\frac{m}{\frac{1}{2}\cdot m + \frac{1}{2} - \max_{\eta_i \in \eta^{\mathsf{S}}} \frac{S_i^{\mathsf{S}}}{BS}} = 2 + \frac{1 - 2\cdot\max_{\eta_i \in \eta^{\mathsf{S}}} \frac{S_i^{\mathsf{S}}}{BS}}{\frac{1}{2}\cdot m + \frac{1}{2} - \max_{\eta_i \in \eta^{\mathsf{S}}} \frac{S_i^{\mathsf{S}}}{BS}},$$

which converges to 2 with an arbitrary large m.

B. Achieving Resource Efficiency without Compromising Timing Guarantees for Multi-Block RT-Blockchain

Now, we also consider the resource efficiency for transactions on a multi-block RT-blockchain, as follows.

Problem Statement: We develop a deadline-based scheduling algorithm *and* a schedulability condition for η^{S} scheduled by the scheduling algorithm on a multi-block blockchain system, such that G1$^{\mathsf{S}}$ in Section IV-C and G2$^{\mathsf{S}}$ are satisfied:

G2$^{\mathsf{S}}$. The number of generated blocks is minimized (achieving *resource efficiency*).

While applying EDF$^{\texttt{WC}}$ associated with Theorems 4 and 5 can achieve G1$^{\mathsf{S}}$, but it is difficult to achieve G2$^{\mathsf{S}}$. Suppose that the highest-priority unassigned transaction job in the waiting queue is not included in one of the m blocks, even though its transaction size is not larger than the unoccupied portion of one of the m blocks (i.e., non-work-conserving deviated from EDF$^{\texttt{WC}}$); then, the theorems do not necessarily guarantee schedulability. This necessitates a mechanism that safely delays transaction jobs to some blocks in later slots, without compromising the schedulability; then, the mechanism helps achieve G2$^{\mathsf{S}}$ whenever there exists a block whose utilization is low under EDF$^{\texttt{WC}}$.

To this end, we develop a lazy algorithm called EDF$^{\texttt{Lazy}}(r)$ as follows, where r is a positive rational number less than m.

Algorithm of EDF$^{Lazy}(r)$ for η^{S} with m blocks: At every beginning of each slot (i.e., $t^s(\texttt{SL}_j)$), sort all transactions in the waiting queue by their absolute slot-unit deadlines. Then, assign each sorted transaction to one of the m blocks one by one using the first-fit policy, until the sum of $\frac{S_i^{\mathsf{S}}}{BS}$ for all assigned transactions is no smaller than r (note that the transaction job which makes the sum no smaller than r is also included in one of the m blocks). Let m' denote the number of blocks occupied by at least one transaction so far. We apply EDF$^{\texttt{WC}}$ for the remaining unassigned transactions as if we have only the m' blocks instead of m.

Then, we derive the same schedulability condition for EDF$^{\texttt{Lazy}}(r)$ as that for EDF$^{\texttt{WC}}$, as follows.

Theorem 6: A set of periodic/sporadic slot-level blockchain transaction tasks η^{S} is schedulable by EDF$^{\texttt{Lazy}}(\texttt{LOAD}(\eta^{\mathsf{S}}))$ on a m-block blockchain system, if Eq. (13) holds.

$$\texttt{LOAD}(\eta^{\mathsf{S}}) \le \texttt{LOAD}^{**}(\eta^{\mathsf{S}}), \tag{13}$$

where $\texttt{LOAD}^{**}(\eta^{\mathsf{S}})$ is defined in Eq. (12).

Proof: Applying the same proof technique as Theorem 2, it suffices to prove the following: in any slot, if the sum of the size of transactions in the waiting queue is larger than $BS \cdot \texttt{LOAD}(\eta^{\mathsf{S}})$, the block can include a set of transactions whose size summation is at least $BS \cdot \texttt{LOAD}(\eta^{\mathsf{S}})$ without violating the priority of transactions (determined by their deadlines).

Since the operation of EDF$^{\texttt{Lazy}}(\texttt{LOAD}(\eta^{\mathsf{S}}))$ is exactly the same as that of EDF$^{\texttt{WC}}$ until the m or smaller blocks include a set of earliest-deadline transactions whose size summation is barely no smaller than $BS \cdot \texttt{LOAD}(\eta^{\mathsf{S}})$, the remaining proof is similar to that of Theorem 5, ■

By its design and Theorem 6, $\text{EDF}^{\texttt{Lazy}}(\texttt{LOAD}(\eta^{\mathsf{S}}))$ is capable of taking both timing guarantees and resource efficiency into consideration. For the former, it provides offline timing guarantees for every job invoked by η^{S} if $\texttt{LOAD}(\eta^{\mathsf{S}}) \leq \texttt{LOAD}(\eta^{\mathsf{S}})$ holds, which is the same degree of timing guarantees as $\text{EDF}^{\texttt{WC}}$. For the latter, it fully utilizes the blocks that should be generated to manage the minimum workload of transaction jobs relevant to timing guarantees. While achieving the former is quantitatively demonstrated by Theorem 6, achieving the latter will be demonstrated in Section VII using real experiments.

VII. Implementation and Evaluation

Implementation. Our RT-Blockchain system implementation is built on Turtlecoin [22], a commercially available and highly extensible blockchain platform. We integrated the PoS consensus mechanism seamlessly into our system. Additionally, we introduced user-defined periods and deadlines for each transaction, allowing users to customize these parameters according to their specific requirements. Nodes are organized into a pool of validators, and the block-generating node is randomly selected during each block time. To align with Ethereum, we set the block time to $BT = 12$ seconds. The size of a single block is set to $BS = 100$K bytes. Within the p2p network, nodes continuously monitor each other at regular intervals to stay updated on new blocks and transactions. If a block-validator's hash calculation does not match that of a new block, the block is rejected from their chain. Final acceptance of a block occurs when a majority (66%) of validators have included the block in their main chain.

The transaction pool was structured as a multi-container system, organized by various indexes enabling sorting based on deadline or time of arrival. At regular intervals corresponding to the block time, the block-generating node selected transactions from the transaction pool to construct blocks, following predefined scheduling rules: FIFO that is the baseline used in most blockchain systems, $\text{EDF}^{\texttt{WC}}$ in Section VI-A, and $\text{EDF}^{\texttt{Lazy}}$ in Section VI-B. The block's hash was computed using the secure SHA-256 (Secure Hash Algorithm 256-bit) algorithm, stored, and then distributed to other nodes.

Case Study. To check the correctness of our multi-block RT-blockchain and investigate its run-time behaviors, we set up the following experimental environments. As the testing environment, we fix the maximum number of generated blocks per slot to 8 and execute for 100 slot units. For each set of slot-level transaction tasks η^{S} to be generated, T_i^{S} is uniformly selected in $[1, 5]$ (integer), D_i^{S} is uniformly selected in $[1, 10]$ (integer), and N_i^{S} is uniformly selected in $[1, 5]$ (integer) if $T_i^{\mathsf{S}} = 1$ (otherwise, $N_i^{\mathsf{S}} = 1$). To investigate how transaction size and number of transactions affect the performance, we generate two types of transaction sets, called Type-A and Type-B. For Type-A, $\texttt{LOAD}(\eta^{\mathsf{S}})$ for each η^{S} changes according to the transaction size. Initially, S_i^{S} for each $\eta_i^{\mathsf{S}} \in \eta^{\mathsf{S}}$ is uniformly

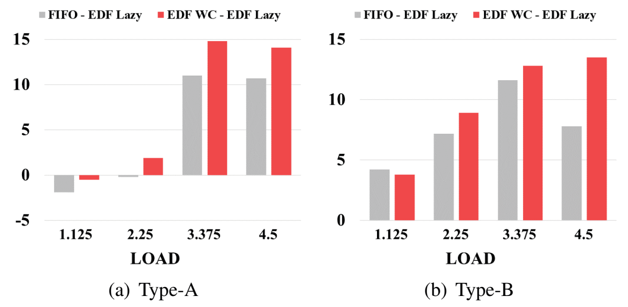


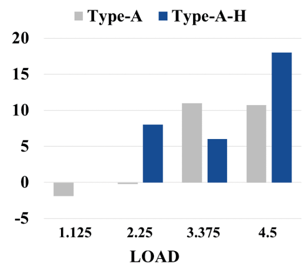
Fig. 4. The number of generated blocks by FIFO, subtracted by that by $\text{EDF}^{\texttt{Lazy}}$ (presented by grey bar), and the number of generated blocks by $\text{EDF}^{\texttt{WC}}$, subtracted by that by $\text{EDF}^{\texttt{Lazy}}$ (presented by red bar)

distributed between 0 and $0.1 \cdot BS$ such that $\texttt{LOAD}(\eta^{\mathsf{S}})$ for η^{S} is close to 1.125. We increase S_i^{S} by multiplying 2, 3, and 4 on S_i^{S} to meet each option of $\left(\max_{\eta_i \in \eta^{\mathsf{S}}} S_i^{\mathsf{S}}\right)$ and generate ten slot-level transaction sets for each options $0.2 \cdot BS$, $0.3 \cdot BS$ and $0.4 \cdot BS$ such that $\texttt{LOAD}(\eta^{\mathsf{S}})$ is close to 2.25, 3.375 and 4.5, respectively. We then construct a user-level transaction task set η from the generated transaction sets corresponding to η^{S} by Theorem 3, such that each of T_i and D_i for $\eta_i \in \eta$ is minimized and each of N_i for $\eta_i \in \eta$ is maximized.

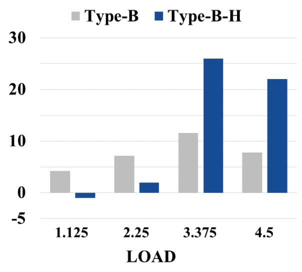
For Type-B, $\texttt{LOAD}(\eta^{\mathsf{S}})$ for each η^{S} is altered by the number of transactions, where the initial option is set to $\left(\max_{\eta_i \in \eta^{\mathsf{S}}} S_i^{\mathsf{S}}\right) = 0.4 \cdot BS$ and $\texttt{LOAD}(\eta^{\mathsf{S}}) \approx 1.125$. The number of transaction tasks in each η^{S} is n-fold increased where n is 2, 3, and 4, such that $\texttt{LOAD}(\eta^{\mathsf{S}})$ is close to 2.25, 3.375, and 4.5, respectively. Each user-level transaction task set is constructed alike the corresponding η^{S} in Type-A.

We have two important observations from Type-A and Type-B scheduled by FIFO, $\text{EDF}^{\texttt{WC}}$ and $\text{EDF}^{\texttt{Lazy}}$ on RT-Blockchain. First, we observe that there is no transaction with a missed deadline under $\text{EDF}^{\texttt{WC}}$ and $\text{EDF}^{\texttt{Lazy}}$. This is the expected result as all the task sets tested satisfy the schedulability condition in Theorems 5 and 6.

Second, $\text{EDF}^{\texttt{Lazy}}$ reduces the number of generated blocks in most cases compared to FIFO, as depicted in Figure 4; later in this section, we will explain why the decrease is in "most" cases rather than "all" cases. As shown in Figure 4(a), for transaction sets with similar $\texttt{LOAD}(\eta^{\mathsf{S}})$ belonging to Type-A, the average decrease in the number of generated blocks by $\text{EDF}^{\texttt{Lazy}}$ compared to FIFO is -1.9, -0.2, 11, and 10.7, respectively each listed in the ascending order of $\texttt{LOAD}(\eta^{\mathsf{S}})$ chosen. When it comes to Type-B illustrated in Figure 4(b), the average decrease is 4.2, 7.2, 11.6, and 7.8, respectively. As observed, neither the increase in the size of transactions (according to Type-A) nor the number of transactions (according to Type-B) has a strong relationship with the performance of $\text{EDF}^{\texttt{Lazy}}$ in terms of reducing the number of generated blocks. Note that since both FIFO and $\text{EDF}^{\texttt{WC}}$ are work-conserving, the difference between the number of blocks generated by FIFO and $\text{EDF}^{\texttt{WC}}$ solely comes from the bin packing order of transactions; therefore, the difference (represented as the gap between the grey and red bar for each $\texttt{LOAD}$ in Figure 4) is not



(a) Type-A vs. Type-A-H



(b) Type-B vs. Type-B-H

Fig. 5. The number of generated blocks by FIFO, subtracted by that by EDF$^{\texttt{Lazy}}$

as significant as the difference between FIFO and EDF$^{\texttt{Lazy}}$.

We further examine which cases might affect the performance of (i.e., the number of generated blocks reduced by) EDF$^{\texttt{Lazy}}$ the most. Since EDF$^{\texttt{Lazy}}$ is made to handle issues of the inefficiency of space that occurs when blocks are generated to fit transactions in blocks at the moment, we generate cases where the transaction periods are harmonic. Specifically, we alter T_i^{S} to 1, 3, and 9 so that a bottleneck is more likely to occur in the blockchain system, generating transaction sets Type-A-H and Type-B-H corresponding to Type-A and Type-B, respectively (where H means Harmonic-period). As illustrated in Figure 5, the sets of harmonic-period transactions (i.e., Type-A-H and Type-B-H) do not consistently lead to a reduction in the number of generated blocks when compared to sets of random-period transactions (i.e., Type-A and Type-B). Hence, asserting the superiority of EDF$^{\texttt{Lazy}}$ in scenarios involving harmonic transaction sets becomes challenging. This underscores that the transaction set's period is not the sole variable influencing performance.

With this insight, we design transaction sets to an extent where EDF$^{\texttt{Lazy}}$ demonstrates exceptional performance (in terms of reducing the number of generated blocks) increase. In FIFO and EDF$^{\texttt{WC}}$, since blocks must be generated to fit all transactions that are released during every block time, it tends to become inefficient when an extra block is made to fit in a single transaction. In Figure 6, we formulate a transaction set with $\texttt{LOAD} = 0.9$, consisting of six Transaction A $(T_i^{\mathsf{S}} = 3, D_i^{\mathsf{S}} = 3, S_i^{\mathsf{S}} = 0.3 \cdot BS, N_i^{\mathsf{S}} = 1)$ and one Transaction B $(T_i^{\mathsf{S}} = 1, D_i^{\mathsf{S}} = 1, S_i^{\mathsf{S}} = 0.3 \cdot BS, N_i^{\mathsf{S}} = 1)$. When scheduled with FIFO (as well as EDF$^{\texttt{WC}}$), three blocks contain all seven transactions during the first slot. Two blocks are filled with three transactions with an idle space of $0.1{\cdot}BS$ and one block is made containing a single transaction, utilizing only $0.3{\cdot}BS$ of space. Additionally, a block utilizing only $0.3{\cdot}BS$ of space is made during the second and third slot units. On the other hand, in the case of EDF$^{\texttt{Lazy}}$, transactions are pushed to the next slots to fully use the empty space that could occur in the next slot units, and hence only one block (because of $3 \cdot 0.3 = \texttt{LOAD} = 0.9 \leq 1.0$) is made in every slot. In this scenario according to the point where every transaction is $0.3 \cdot BS$ and $\texttt{LOAD} \leq 1.0$, EDF$^{\texttt{Lazy}}$ outperforms FIFO (as well as EDF$^{\texttt{WC}}$) by 40 percent in terms of the number of

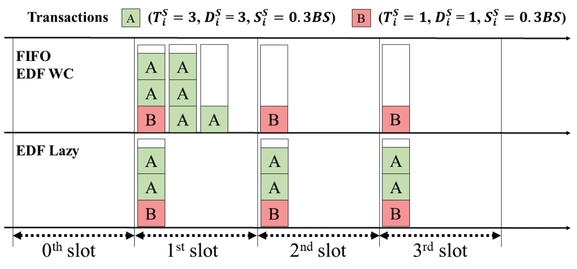

Transactions
A $(T_i^S = 3, D_i^S = 3, S_i^S = 0.3BS)$
B $(T_i^S = 1, D_i^S = 1, S_i^S = 0.3BS)$
FIFO
EDF WC
EDF Lazy
A
B
0th slot
1st slot
2nd slot
3rd slot

Fig. 6. Block generation behaviors by FIFO, EDF$^{\texttt{WC}}$ and EDF$^{\texttt{Lazy}}$

generated blocks, i.e., 5 blocks generated by FIFO versus 3 blocks generated by EDF$^{\texttt{Lazy}}$, per every 3 slot units.

However, when the number of transactions for this specific transaction set is tripled (i.e., 18 Transaction A and 3 Transaction B), such that 21, 3, and 3 blocks each of whose size is $0.3{\cdot}BS$ are released during the 0th, 1st, and 2nd slot, FIFO and EDF$^{\texttt{Lazy}}$ make an equal amount of blocks. FIFO (as well as EDF$^{\texttt{WC}}$) makes 7, 1, and 1 blocks which is a total of 9 blocks for three slot units, and EDF$^{\texttt{Lazy}}$ generates 3 blocks for every slot unit which in every three slot units is the same as FIFO. Hence, the augmentation in performance for EDF$^{\texttt{Lazy}}$ is not solely contingent upon the quantity, dimension or period of transactions, or the value attributed to $\texttt{LOAD}$. As EDF$^{\texttt{Lazy}}$ postpones transactions to an extent where it is stored in a block at the latest slot-unit while keeping the deadline, there are cases where this method negatively affects the performance in terms of the number of generated blocks reduced. This observation is further substantiated by the data illustrated in Figures 4 and 5, e.g., the case for Type-A with $\texttt{LOAD} = 1.125$.

Evaluation of Schedulability Tests. We evaluate the effectiveness of the proposed schedulability tests for periodic/sporadic slot-level blockchain transaction tasks η^{S}: (i) Theorem 4, denoted by $\texttt{LOAD}^{\star}$, and (ii) its improved version Theorem 5 (or equivalently Theorem 6), denoted by $\texttt{LOAD}^{\star\star}$. We target the same multi-block RT-blockchain as that for the case study, in which the block time is $BT = 12$ seconds and the number of maximum generating blocks in a slot is $m = 8$. We consider the following three parameters of slot-level transaction task sets: (P1) the number of slot-level transaction tasks in each task set, determined as 4, 6, 8, 10, or 12, (P2) the maximum transaction size within each task set $\big(\max_{\eta_i \in \eta^{\mathsf{S}}} S_i^{\mathsf{S}}\big)$, determined as $0.3{\cdot}BS$, $0.6{\cdot}BS$, or $0.9{\cdot}BS$, and (P3) the range of $\texttt{LOAD}(\eta^{\mathsf{S}})$, determined as $[0.0m, 0.2m)$, $[0.2m, 0.4m)$, $[0.4m, 0.6m)$ or $[0.6m, 0.8m)$. For each combination of P1, P2 and P3, we randomly generate 200 slot-level transaction task sets, yielding $200 \cdot 5 \cdot 3 \cdot 4 = 12,000$ slot-level transaction task sets in total.

Figure 7(a), (b), and (c) illustrates the ratio of task sets schedulable by $\texttt{LOAD}^{\star}$ and $\texttt{LOAD}^{\star\star}$ according to P1, P2 and P3, respectively. While the schedulable ratio decreases as the number of tasks in each task increases shown in Figure 7(a), $\texttt{LOAD}^{\star\star}$ guarantees 17.8–25.1% more schedulability of transactions compared to $\texttt{LOAD}^{\star}$. In Figure 7(b), we observe that the schedulable ratio decreases but the relative schedu-

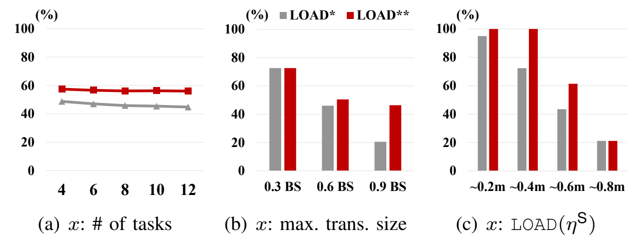


Fig. 7. The ratio of task sets schedulable by $\mathtt{LOAD}^{\star}$ and $\mathtt{LOAD}^{\star\star}$

lability performance of $\mathtt{LOAD}^{\star\star}$ over $\mathtt{LOAD}^{\star}$ increases up to 225.8%, as the maximum transaction size within each task set increases. As we expected, the schedulable ratio decreases as $\mathtt{LOAD}(\eta^{\mathsf{S}})$ increases shown in Figure 7(c). However, the relative performance of $\mathtt{LOAD}^{\star\star}$ over $\mathtt{LOAD}^{\star}$ is maximized when $\mathtt{LOAD}(\eta^{\mathsf{S}})$ is in $[0.2m, 0.4m)$ and $[0.4m, 0.6m)$ (i.e., 138.2% and 141.3%, respectively); this is because the timing guarantees for the case of $[0.0m, 0.2m)$ and $[0.6m, 0.8m)$ are respectively too easy and too difficult, making the schedulability performance of $\mathtt{LOAD}^{\star\star}$ similar to that of $\mathtt{LOAD}^{\star}$.

VIII. Related Work

From the outset, the transaction rate and latency of blockchain systems have been in the attention of many researchers [4] [5] [6]. Thus, enhancing the transaction rate of blockchain systems has been at the scope of many researchers. [23] presents Conflux, a decentralized and smart-contract-enabled blockchain system with high throughput and fast confirmation, operating with a novel consensus protocol and critical optimizations. [24] with their Bitcoin-NG increases the transaction rate by enlarging the block size and avoids numerous forking issues by addressing a new consensus. [25] introduces a Permissioned Blockchain Framework (PBF) that surpasses Bitcoin-derived blockchains in terms of throughput and latency while upholding the same trust assumptions. Efforts to minimize latency and increase transaction capacity in blockchains have led to the development of scaling techniques like sharding [26] [27] [28] and layer-2 solutions such as rollups [29]. However, neither of these studies achieved nor focused on guaranteeing deadlined tasks.

Recent research has focused on blockchain systems handling transactions with deadlines. This is particularly important for payment channel techniques addressing blockchain transaction scalability [30] [31]. Notable contributions include the novel transaction ordering method proposed by Shi et al. in [32], which employs reinforcement learning to optimize block parameters. Despite these advancements, none has yet managed to process transactions to meet deadlines without missing under strict real-time requirements.

IX. Discussion

While the proposed RT-blockchain offers timing guarantees for transactions, several considerations need to be addressed when applying it to a public chain, which have not been discussed in the previous sections.

Network. The presence of network latency can result in validators perceiving different chain heads. To address this issue, Ethereum has implemented the LMD-GHOST [33] algorithm, which can also be adopted in the RT-blockchain to mitigate such concerns.

Security. The RT-blockchain inherits security issues from PoS, such as finality delay, double finality, and reorganization attacks [34]. However, it has been proven that the success rate of these attacks is low, and therefore, they are not considered critical in the context of RT-blockchain. Nevertheless, as RT-blockchain introduces a novel design element in the form of a deadline-based prioritization policy, it may introduce new security vulnerabilities that need to be considered. For example, an attacker may be able to perform a DoS attack by sending a large number of transactions with urgent deadlines to prevent the processing of transactions from others. To mitigate this attack, transaction fees can be implemented, requiring higher transaction fees for transactions with shorter deadlines. Moreover, malicious block-generating nodes may intentionally delay block propagation and construct blocks with transactions not in deadline order. To address this issue, the RT-blockchain can establish slashing conditions to incentivize validators to adhere to the protocol.

Scaling. It is worth noting that the scheduling principles proposed in this paper are general enough to be applicable to other consensus algorithms, beyond just PoS. Additionally, it can be applied to private and hybrid chains [35] where performance and timing guarantees for transactions are of greater significance. In such chains, our approach could be more easily adopted by limiting only authorized users to send deadline-based transactions. Scaling can be considered to enhance the performance of the RT-Blockchain. On-chain and off-chain scaling methods, such as sharding and layer-2 rollups, can present promising results. Nonetheless, given the RT-Blockchain's primary emphasis on ensuring precise timing guarantees for each transaction, the straightforward application of scaling techniques is not suitable. Instead, a tailored approach for the implementation of each method is necessary to align with the stipulated requirements presented in this paper.

X. Conclusion

This paper introduces a blockchain system that takes into account transaction deadlines, enabling timing guarantees of periodic transactions. Our system architecture is specifically designed to fully accommodate real-time scheduling by incorporating deadlines as a criterion for selecting transactions. Additionally, it focuses on optimizing space efficiency by dynamically generating an appropriate number of blocks per block time. By implementing real-time scheduling principles, the integration of periodic transactions into our blockchain system becomes seamless. This novel approach introduces exciting prospects for time-sensitive applications, significantly enhancing the overall efficiency and usability of blockchain technology in real-time scenarios.

Acknowledgement

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (2022R1A4A3018824, 2022K2A9A1A01097764, 2022R1F1A1074495, RS-2023-00248143), and Institute of Information & Communications Technology Planning & Evaluation (IITP) (2022-0-00688, AI Platform to Fully Adapt and Reflect Privacy-Policy Changes, 2022-0-01199). We thank Prof. Nan Guan at City University of Hong Kong, for his advice. We also thank the shepherd and reviewers for their valuable comments and suggestions.

References

[1] B. K. Mohanta, S. S. Panda, and D. Jena, "An overview of smart contract and use cases in blockchain technology," in *2018 IEEE 9th international conference on computing, communication and networking technologies (ICCCNT)*, pp. 1–4.

[2] P. Jittrapirom, V. Caiati, A. M. Feneri, S. Ebrahimigharehbaghi, M. J. Alonso-González, and J. Narayan, "Mobility as a service: A critical review of definitions, assessments of schemes, and key challenges," *Urban Planning*, vol. 2, no. 2, pp. 13–25, 2017.

[3] M. Baron, J. Brogaard, B. Hagströmer, and A. Kirilenko, "Risk and return in high-frequency trading," *Journal of Financial and Quantitative Analysis*, vol. 54, no. 3, pp. 993–1024, 2019.

[4] R. Yasaweerasinghelage, M. Staples, and I. Weber, "Predicting latency of blockchain-based systems using architectural modelling and simulation," in *2017 IEEE International Conference on Software Architecture (ICSA)*, pp. 253–256.

[5] M. Kuzlu, M. Pipattanasomporn, L. Gurses, and S. Rahman, "Performance analysis of a hyperledger fabric blockchain framework: throughput, latency and scalability," in *2019 IEEE international conference on blockchain (Blockchain)*, pp. 536–540.

[6] C. Wang and N. Raviv, "Low latency cross-shard transactions in coded blockchain," in *2021 IEEE International Symposium on Information Theory (ISIT)*, pp. 2678–2683.

[7] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized business review*, 2008.

[8] M. Jakobsson and A. Juels, "Proofs of work and bread pudding protocols," in *Secure Information Networks: Communications and Multimedia Security IFIP TC6/TC11 Joint Working Conference on Communications and Multimedia Security (CMS'99)*. Springer, 1999, pp. 258–272.

[9] F. Saleh, "Blockchain without waste: Proof-of-stake," *The Review of financial studies*, vol. 34, no. 3, pp. 1156–1190, 2021.

[10] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger."

[11] E. Kyriakides and M. Polycarpou, *Intelligent monitoring, control, and security of critical infrastructure systems*. Springer, 2014, vol. 565.

[12] H. Stadtler, "Supply chain management: An overview," *Supply chain management and advanced planning: Concepts, models, software, and case studies*, pp. 3–28, 2014.

[13] S. Baruah, A. Mok, and L. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *1990 IEEE Real-Time Systems Symposium (RTSS)*, pp. 182–190.

[14] Q. Wang, R. Li, Q. Wang, and S. Chen, "Non-fungible token (NFT): Overview, evaluation, opportunities and challenges," *arXiv preprint arXiv:2105.07447*, 2021.

[15] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, and Y. Liu, "A survey on the scalability of blockchain systems," *IEEE Network*, vol. 33, no. 5, pp. 166–173, 2019.

[16] H. Dang, T. T. A. Dinh, D. Loghin, E.-C. Chang, Q. Lin, and B. C. Ooi, "Towards scaling blockchain systems via sharding," in *Proceedings of the 2019 International Conference on Management of Data*, pp. 123–140.

[17] J. Poon and T. Dryja, "The bitcoin lightning network," *Scalable o-chain instant payments*, 2015.

[18] C. Sguanci, R. Spatafora, and A. M. Vergani, "Layer 2 blockchain scaling: A survey," *arXiv preprint arXiv:2107.10881*, 2021.

[19] A. Singh, K. Click, R. M. Parizi, Q. Zhang, A. Dehghantanha, and K. K. R. Choo, "Sidechain technologies in blockchain networks: An examination and state-of-the-art review," *Journal of Network and Computer Applications*, 2020.

[20] J. Poon and V. Buterin, "Plasma: Scalable autonomous smart contracts," *White paper*, pp. 1–47, 2017.

[21] L. T. Thibault, T. Sarry, and A. S. Hafid, "Blockchain scaling using rollups: A comprehensive survey," *IEEE Access*, 2022.

[22] "Turtlecoin," http://www.turtlecoin.info/, accessed: 2023-05-26.

[23] C. Li, P. Li, D. Zhou, Z. Yang, M. Wu, G. Yang, W. Xu, F. Long, and A. C.-C. Yao, "A decentralized blockchain with high throughput and fast confirmation," in *2020 USENIX Annual Technical Conference (ATC)*, pp. 515–528.

[24] J. Göbel and A. E. Krzesinski, "Increased block size and bitcoin blockchain dynamics," in *2017 27th International Telecommunication Networks and Applications Conference (ITNAC)*, pp. 1–6.

[25] X. Min, Q. Li, L. Liu, and L. Cui, "A permissioned blockchain framework for supporting instant transaction and dynamic block size," in *2016 IEEE Trustcom/BigDataSE/ISPA*, pp. 90–96.

[26] H. Dang, T. T. A. Dinh, D. Loghin, E.-C. Chang, Q. Lin, and B. C. Ooi, "Towards scaling blockchain systems via sharding," in *Proceedings of the 2019 international conference on management of data*, pp. 123–140.

[27] C. Huang, Z. Wang, H. Chen, Q. Hu, Q. Zhang, W. Wang, and X. Guan, "Repchain: A reputation-based secure, fast, and high incentive blockchain system via sharding," *IEEE Internet of Things Journal*, vol. 8, no. 6, pp. 4291–4304, 2020.

[28] P. Zheng, Q. Xu, Z. Zheng, Z. Zhou, Y. Yan, and H. Zhang, "Meepo: Sharded consortium blockchain," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pp. 1847–1852.

[29] L. T. Thibault, T. Sarry, and A. S. Hafid, "Blockchain scaling using rollups: A comprehensive survey," *IEEE Access*, 2022.

[30] V. Sivaraman, S. B. Venkatakrishnan, M. Alizadeh, G. Fanti, and P. Viswanath, "Routing cryptocurrency with the spider network," in *Proceedings of the 2018 17th ACM Workshop on Hot Topics in Networks*, pp. 29–35.

[31] U. Goel, R. Sonanis, I. Rastogi, S. Lal, and A. De, "Criticality aware orderer for heterogeneous transactions in blockchain," in *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pp. 1–4.

[32] J. Shi, H. Wu, D. Luo, H. Gao, and W. Zhang, "Instantchain: Enhancing order-execute blockchain systems for latency-sensitive applications," in *International Conference on Database Systems for Advanced Applications*. Springer, 2023, pp. 483–498.

[33] V. Buterin, D. Hernandez, T. Kamphefner, K. Pham, Z. Qiao, D. Ryan, J. Sin, Y. Wang, and Y. X. Zhang, "Combining ghost and casper," 2020.

[34] Ethereum, "Ethereum proof-of-stake attack and defense," https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/attack-and-defense/, 2023.

[35] G. Sagirlar, B. Carminati, E. Ferrari, J. D. Sheehan, and E. Ragnoli, "Hybrid-IoT: Hybrid blockchain architecture for internet of things-pow sub-blockchains," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pp. 1007–1016.