Context-based Smart Contracts For Appendable-block Blockchains

Henry C. Nunes*, Roben C. Lunardi*†, Avelino F. Zorzo*, Regio A. Michelin‡§ and Salil S. Kanhere‡
*Pontifical Catholic University of Rio Grande do Sul (PUCRS) - Porto Alegre, Brazil
†Federal Institute of Rio Grande do Sul (IFRS) - Porto Alegre, Brazil
‡University of New South Wales (UNSW) - Sydney, Australia
§Cyber Security CRC (CSCRC) - Sydney, Australia
E-mail: {henry.nunes,roben.lunardi}@edu.pucrs.br, avelino.zorzo@pucrs.br,
regio.michelin@cybersecuritycrc.org.au, salil.kanhere@unsw.edu.au

***Abstract*—Currently, blockchain proposals are being adopted to solve security issues, such as data integrity, resilience, and non-repudiation. To improve certain aspects, *e.g.*, energy consumption and latency, of traditional blockchains, different architectures, algorithms, and data management methods have been recently proposed. For example, appendable-block blockchain uses a different data structure designed to reduce latency in block and transaction insertion. It is especially applicable in domains such as Internet of Things (IoT), where both latency and energy are key concerns. However, the lack of some features available to other blockchains, such as Smart Contracts, limits the application of this model. To solve this, in this work, we propose the use of Smart Contracts in appendable-block blockchain through a new model called context-based appendable-block blockchain. This model also allows the execution of multiple smart contracts in parallel, featuring high performance in parallel computing scenarios. Furthermore, we present an implementation for the context-based appendable-block blockchain using an Ethereum Virtual Machine (EVM). Finally, we execute this implementation in four different testbed. The results demonstrated a performance improvement for parallel processing of smart contracts when using the proposed model.**

***Index Terms*—IoT, Blockchain, context-based, smart contracts.**

I. Introduction

Distributed applications, such as distributed databases [1], have existed for a long time. However, they are dependent on an assumption of trust, *i.e.* nodes that compose the environment are honest. Alternatively, trust can be delegated to a third party that can assure that the environment is trustable. One such third party is a certificate authority [2] that can assure the identity of the nodes. The blockchain concept changed this scenario, ensuring the robust execution of a deployed application even if some of the participating nodes misbehave or become unavailable. Blockchain also offers other benefits, including audibility, transparency, and the possibility to have decentralized applications (dApps) [3]. The combination of those benefits allows to use blockchain in multiple domains, such as financial operations [4], supply chain [5], Internet of Things (IoT) [6] [7], health care [8], smart vehicles [9], smart cities [10] [11] and others [12].

However, a number of challenges need to be addressed before the widespread uptake of this technology. Zheng *et al.* [4] highlight several such challenges, including scalability and privacy leakage. Furthermore, two key challenges that are specifically relevant to IoT applications are high latency [13] and high energy consumption from some consensus algorithms [14]. They are relevant because of the high amount of generated data, the need for low latency for specific applications, and the hardware constraints on most IoT devices.

SpeedyChain [7] [11] [15], which uses the appendable-block blockchain model, was specifically proposed to address these issues by employing a different block structure and network architecture. The block structure allows the insertion of multiple transactions in the blockchain at the same time, and the network architecture uses the concept of gateways that allows higher performance devices to process transactions in the blockchain [7]. Moreover, it uses the practical byzantine fault tolerance (PBFT) consensus algorithm for achieving energy efficiency [14]. Michelin *et al.* [11] demonstrated promising performance improvements by adopting SpeedyChain for storing and managing sensor data collected from smart vehicles in the context of smart city applications.

Smart contracts can benefit from the appendable-blocks blockchain model used in SpeedyChain. In specific scenarios, the insertion of transactions in parallel can increase the performance of smart contracts. Moreover, the addition of smart contracts gives flexibility to applications that can work on top of SpeedyChain. For example, smart contracts can be used to help in IoT security in the following ways [6]: (*i*) providing an IoT update service; (*ii*) allowing a marketplace between devices; (*iii*) management and control of an IoT network; and (*iv*) delegated processing and workload balancing.

Due to those benefits, in this work, we propose a model

This paper was achieved in cooperation with HP Brasil using incentives of Brazilian Informatics Law (Law n 8.248 of 1991). This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. Avelino F. Zorzo is supported by CNPq (315192/2018-6) and FAPERGS. This work was supported by the INCT Forensic Sciences through the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq − process # 465450/2014-8). Also, we thank the support from IFRS and Cyber Security Research Centre Limited whose activities are partially funded by the Australian Government's Cooperative Research Centres Programme.

978-1-7281-6680-3/20/$31.00 ©2020 IEEE

Authorized licensed use limited to: Pontificia Universidade Catolica do Rio Grande do Sul (PUC/RS). Downloaded on August 04,2021 at 13:12:16 UTC from IEEE Xplore. Restrictions apply.

to provide the smart contracts capability on the SpeedyChain architecture, called context-based for appendable-block blockchain. The model works by isolating a group of smart contracts - from the same context - in a single block. Each group can receive transactions independently - in parallel - to the other groups, thus increasing performance. To validate our model, we present an evaluation focused on delegated processing for route calculation based on GPS data for IoT devices. Although we apply the context-based smart contracts model to one specific blockchain technology, the ideas presented in this paper can be generalized and used in different architectures.

II. Background

This section presents the background required to understand the model of smart contracts for appendable-block blockchain. Therefore, we discuss different concepts used to build the model proposed in this paper. First, we present the functions used in the paper, followed by the immutable-block blockchain structure, the main appendable-block blockchain concepts, and the adopted smart contracts definition.

A. Mathematical functions

The functions we use throughout these paper are:

- We will use function p to extract element e from a tuple using a lambda function as presented in equation 1:

$$p_e(tuple) = (\lambda(T_1, ..., T_e, ...T_n) \to T_e) \tag{1}$$

  As an example for (1), considering $t = (1, 6, 3)$ the operations $p_1(t), p_2(t)$ and $p_3(t)$ will result in $1, 6$ and 3 respectively.
- We will use $H(x)$ as a hash function that can receive any sequence of bits x as input and outputs another sequence of bits [16]. The specific $H(x)$ function used here will be abstracted. Properties of a good hash function, as collision-free, pseudo-randomness and unpredictability will just be assumed as true.
- The PK function receives a digital signature as input and returns the public key from an asymmetric cryptography scheme. For this work, we consider a cryptography scheme in which you can recover a public key directly from a digital signature [17].

B. Immutable-Block Blockchain

A blockchain is a distributed ledger that permanently stores all transactions that brought the system to the current state [18]. Transactions are stored in blocks, once a block is added to the blockchain it is immutable, hence we call this type of blockchain an immutable-block blockchain. It is distributed because the system will work based on a Peer-to-Peer (P2P) network [19], in which each node in the blockchain network will maintain a local copy of the blockchain.

The system state is changed by one node using the consensus algorithm to add new blocks. That causes all nodes to change their local copy of the system state. The consensus algorithm works as a pre-agreement of how the system can progress, it helps all nodes to converge to the same system state despite some nodes malfunctioning or acting in a malicious way [20]. There are multiple consensus algorithms, like Proof of Work (PoW) [18], Proof of Stake (PoS) [21] and Practical Byzantine Fault Tolerance (PBFT) [22]. The consensus algorithm will be abstracted in this work to the function $performConsensus$ that returns $true$ if a proposed block is authorized to be added to the blockchain or $false$ otherwise. More details about consensus are discussed in [3].

The behavior of individual nodes is not presumed as correct or honest. The consensus algorithm guarantees that even if part of the nodes work maliciously or incorrectly, the data inserted in the blockchain can be trusted, and the system will work correctly. This feature is one of the major benefits of the blockchain. There are other benefits such as auditability [23], since all transactions are stored as a ledger the current state can be audited at any time; resilience [23], since a blockchain is distributed in a P2P network, hence, if any node fails, the blockchain can continue to work.

C. Appendable-Block Blockchain

The immutable-block blockchain is nowadays the most used data architecture in blockchains. It is used in important data ledger technologies such as Ethereum [24], Bitcoin [18] and Hyperledger Fabric [25]. However, there are other architectures proposed by industry, like the Tangle architecture proposed by IOTA [26], and by academia, as SpeedyChain [15], whose data structure is relevant to this work.

SpeedyChain is designed for the context of IoT, where devices usually have low computing power and limited storage capacity. This limits the capability to use a blockchain because of the necessity to store the blockchain in the nodes and the computing power required for some consensus algorithms such as PoW. Furthermore, high communication latency is a key factor in some IoT applications, which is another limiting factor in the use of blockchain in this case. To mitigate these problems, SpeedyChain proposes: (*i*) to use gateways with more processing power and storage to work as blockchain full nodes, while other IoT devices have to connect to those gateways to access the blockchain. This removes the burden of maintaining a full node for limited IoT devices; and, (*ii*) a mutable blocks architecture, referenced as appendable-block blockchain, where blocks can be expanded with new transactions. This approach allows the blockchain to expand appending transaction in multiple blocks in parallel, while immutable-block blockchain can insert new transactions just by the introduction of a new block [7] [11] [15]. For more details about SpeedyChain architecture and security aspects, please refer to [15].

Formally, a generalization of the appendable-block blockchain architecture has a set of n nodes $N = \{N_1, ..., N_n\}$. Nodes are gateways and IoT devices that generate data. Each $\{N_i\}$ has a pair of public/secret keys (NPK_i, NSK_i) respectively from an asymmetric cryptography scheme, where the public keys are accessible to every participant of the blockchain [15].

Authorized licensed use limited to: Pontificia Universidade Catolica do Rio Grande do Sul (PUC/RS). Downloaded on August 04,2021 at 13:12:16 UTC from IEEE Xplore. Restrictions apply.

The data structure in an appendable-block blockchain is a non-empty set BC of blocks. Each block is a tuple (BH, BL). BL, named block ledger, is a set of transactions that can be incremented as necessary and linked to a BH. The BH, named block header, is another tuple composed of $(ParentHash, NPK_i, T)$, with meta-data about the block:

- $ParentHash$ is the result of $H(BH)$ of the block inserted in the blockchain, *i.e.* i-1. It works as a pointer to the previous block.
- NPK_i is the public key of a member of the set N, only one block can have a specific NPK_i. To enforce that the $\nexists x \mid x,\, b \in BC \wedge p_2(p_1(x)) = p_2(p_1(b))$ post-condition must be respected. The node that has the NSK_i to the NPK_i of a block is said to be the block owner, and only that node can append new transactions to that block.
- T is the first transaction inserted into a block and the only one to be part of the block header, furthermore, this transaction is the first transaction signed by a pair of public/secret keys, the public key is the NPK_i value.

The BC set will form a hash-linked list of blocks B connected by their $ParentHash$ in the BH.

A transaction withholds data generated by the nodes, the data content depends on the application and context. In the appendable-block blockchain a transaction is represented as a tuple of $(Data, PT, Sig)$, where: $Data$ is specific to the node generating data through the creation of the transaction; PT is the hash of the previous transaction inserted into the block, it works as a hash-link connecting the transactions in the block. If it is the first transaction in the block, then it will refer to the hash of the BH; and Sig is a digital signature from the node originating the $Data$ [15].

Before presenting how appendable-block blockchain adds new blocks and appends transactions, we present the auxiliary functions $newBlock$ as shown in Equation 2, which summarizes the creation of a new block filling the necessary fields, and function lB (Equation 3), which returns a block that has no other block with the $ParentHash$ in the header pointing to it. In practice, it is the last block created in the blockchain.

$$\begin{aligned} &newBlock(T, BC) = \\ &((H(p_1(lB(BC))), PK(p_3(T)), T), \{\}) \end{aligned} \tag{2}$$

$$\begin{aligned} &lB(BC) = x | x \in BC \wedge \\ &(\nexists y \in BC \wedge p_1(p_1(y)) = H(p_1(x)) \wedge x \neq y) \end{aligned} \tag{3}$$

To add a new block to the blockchain, function $addB(BC, T)$ (Equation 4) is used. It creates a new block for a node and appends the new block to the blockchain if there is no other block with the same NPK as the transaction signature public key $PK(p_3(T))$. The predicate $uniqueBlock$ (Equation 5) guarantees this requirement.

$$addB(BC,T) = \begin{cases} BC \cup newBlock(T, BC), \\ \text{if } uniqueBlock \\ \\ BC, \text{ otherwise} \end{cases} \tag{4}$$

$$uniqueBlock = \nexists x . x \in BC \wedge p_2(p_1(x)) = PK(p_3(T)) \tag{5}$$

New transactions are generated by nodes with new $Data$ to be inserted into the blockchain. This operation is performed only if the node's public key (NPK_i) is present in a block header BH. Function $appendT$ (Equation 6) specifies the insertion of a new transaction T in a block B that has a public key equal to the public key used in the transaction signature.

$$appendT(BC,T) = \begin{cases} (BC - B) + updateB(B, T), \\ \text{if } p_2(T) = PreTHash(B) \\ \\ BC, \text{otherwise} \end{cases} \tag{6}$$

$updateB$ (Equation 7) is an auxiliary function to generate an updated block where transaction T is appended. Function $PreTHash$ (Equation 8) returns the hash of a previous transaction appended to the block or the block B header hash. This hash will be used to check if a transaction is pointing to the PT field of the last transaction inserted into the block.

$$updateB(B, T) = (p_1(B), p_2(B) \cup T) \tag{7}$$

$$PreTHash(B) = \begin{cases} H(p_1(B)), \text{ if } |p_2(B)| = 0 \\ H(lastT(B)), \text{otherwise} \end{cases} \tag{8}$$

Algorithm 1 shows how the main operation works on this model for the insertion of new transactions in a continuous way. The $mempool$ consists of a set of transactions submitted to the blockchain by multiple nodes, but not yet appended to the blockchain, this $mempool$ is shared by all nodes. Function $poll$, on the $mempool$, returns one transaction of the set. Before processing a new transaction, it checks if a block with the public key of the signer exists through function $exists$ (line 6); if not, a new block is processed by the consensus algorithm (line 7) and if approved, a new block is inserted and broadcast to the network (lines 9 and 10). Otherwise, the proposed transaction is processed by the consensus algorithm (line 12) and if accepted, it is appended to the block owned by the transaction signer (more details about this algorithm is described in [15]).

Algorithm 1 Main operation for appendable-block blockchain

```
1  Result: BC //Updated state
2  Input: mempool, BC //Original state
3
4  while(True)
5   T = poll(MemPool)
6   if(!exists(PK($p_3$(T))))
7     ConsensusResponse = performConsensus(B)
8     if(ConsensusResponse)
9         broadcast(B)
10        BC = addB(BC, T)
11  else
12    ConsensusResponse = performConsensus(T)
13    if(consensusResponse)
14      broadcast(T)
15      BC = appendT(BC, T)
```

Authorized licensed use limited to: Pontificia Universidade Catolica do Rio Grande do Sul (PUC/RS). Downloaded on August 04,2021 at 13:12:16 UTC from IEEE Xplore. Restrictions apply.

D. Smart contracts

There are different models for smart contracts implementation in blockchains. One of these models is the one used by Ethereum [27]. In Ethereum, smart contracts are stored in the blockchain as special transactions. Those transactions are bytecode that can be processed by the Ethereum Virtual Machine (EVM). Each node in the Ethereum network has an EVM. Calls, like a program call, for a smart contract, are appended to the blockchain as transactions. The transaction contains the bytecode, representing the program call, to be processed in the context of a specific smart contract.

It is important to evaluate the performance of smart contracts in the blockchain. To this end, several works have been proposed. Aldweesh *et al.* proposed in their works [28] [29] to analyze the computational performance of EVM's operation codes, and compare the results to the monetary incentives miners receive from their work. The Hyperledger project proposed a more general approach [30], which presents guidelines for performance evaluation of any blockchain.

Throughout this work, we will refer to a generic virtual machine as a function VM, which works similarly to the EVM [24], *i.e.*, $VM(S, Data) = S'$. VM receives two inputs: $Data$, which is the bytecode with operations for the virtual machine; and, S, which is a pointer to a data structure that contains a state for the virtual machine and multiple smart contracts. The output for the VM function is a reference to new state S', based on the modifications that the $Data$ incurred. We store different states in a Merkle Patricia Trie [27]. A new S with no modifications is refereed as the constant $newS$.

III. Context-based Model

This section presents the context-based model for extending the appendable-block blockchain architecture with the smart contracts capability. In this model, each block can hold a block state, which is a data structure capable of holding a mapping from an address to a smart contract state and code. Remember from the previous section that we use as a reference for the data structure a Merkle Patricia Trie [27]. Aside from these new blocks with this capability to hold the block state, the blocks that carry just data from nodes, as presented in appendable-block blockchain (see Section II-C) still exist. During the creation of a new block it is decided which type of block will be created: a pure data block, which will carry just data; or, a block with context, which will hold a state.

Figure 1 presents an overview of the model. In the figure, three blocks are presented, only blocks $B-1$ and $B+1$ hold smart contracts, identified by the absence of a block PK_i. Therefore they are blocks with context. While block B is a pure data block, identified by the presence of a PK_i. A context can have smart contracts, those smart contracts are isolated from other block contexts and can only interact with smart contracts in the same block, thus this model is called context-based model. In Figure 1, Smart Contract I can interact with Smart Contract II and III, this includes making a call to the other smart contracts, changing their states and querying information. Smart Contract IV, in block $B+1$, cannot interact with Smart Contract I, II and III in any form. A blockchain that adopts the proposed model can have any amount of blocks with context. The transactions stored in those blocks will carry a bytecode that represents a call for a smart contract or a command to create a new smart contract.

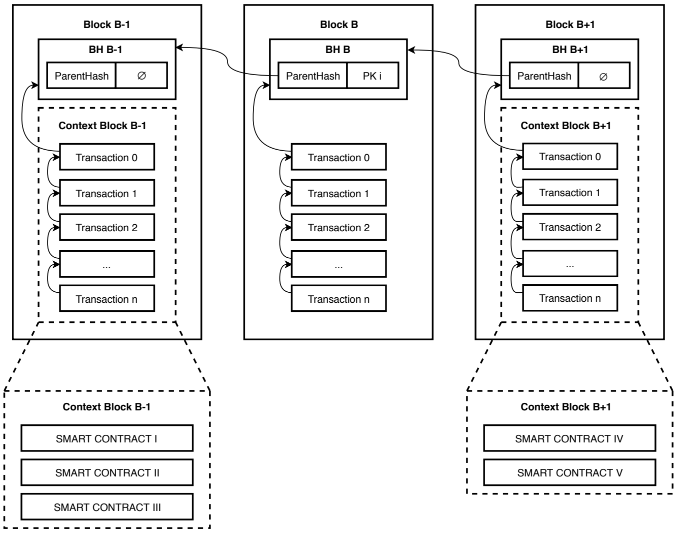

Fig. 1. Smart contracts and block context

All the functions in an appendable-block blockchain work exactly as previously presented in Section II-C, unless stated otherwise. The elements of the tuple (BH, BL) representing the blockchain are different. BH is a tuple $(ParentHash, Index, NPK, CTransaction)$ where $ParentHash$ and NPK work as in the appendable-block architecture, although when the block has a context the value of NPK will be equal to $\emptyset$, because there is no block owner or restriction of devices who can operate in a block. The $Index$ field is a natural number whose value corresponds to the order of blocks created in the blockchain. The $CTransaction$ field stores a **committed transaction**, it is different from a **transaction**. A node can check if a block B has a context by using function $HasContext$ (Equation 9), and get a specific block by the index using function $getBlock$ (Equation 10).

$$HasContext(B) = \begin{cases} true & \text{if } P_3(P_1(B)) = \emptyset \\ false & \text{if } P_3(P_1(B)) \neq \emptyset \end{cases} \quad (9)$$

$$getBlock(Index) = x | x \in BC \wedge P_2(x) = Index \quad (10)$$

A **committed transaction** is defined as a tuple $(Data, Sig, PT, BlockState)$, it is originated from a transaction that was processed by a node. The tuple fields are:

- $Data$ is a binary sequence that depending on the block type will be treated differently. If it is a pure data block, then $Data$ represents data generated by a node, which will not be processed as *bytecode*. If it is a block with a block state, then it is a *bytecode* that will be inserted as an input in the VM function.

Authorized licensed use limited to: Pontificia Universidade Catolica do Rio Grande do Sul (PUC/RS). Downloaded on August 04,2021 at 13:12:16 UTC from IEEE Xplore. Restrictions apply.

- Sig is the digital signature of the transaction that generated this committed transaction.
- PT is the hash value of the previous inserted committed transaction or the block header;
- $BlockState$ is a pointer to a data structure holding the block state, as the Merkle Patricia Trie. The last committed transaction in the block has the most updated state, its value is generated by VM.

When a $Data$ transaction includes bytecode to a block with context, it will be processed by function VM (Equations 11). For that, $Data$ and $BlockState$ of the last inserted committed transaction in the block are inserted as input in the VM function, the resulting pointer to a new state will be attached to a new committed transaction in the $BlockState$ field. S' is a pointer to the resulting state, S is the original BlockState and $NewS$ is an empty BlockState.

$$VM(S, Data) = \begin{cases} S' & \text{if } S \neq \emptyset \\ VM(NewS, Data) & \text{if } S = \emptyset \end{cases} \tag{11}$$

A node that wants to operate on the blockchain will create a transaction for that operation. The transaction is composed of $(Data, ToBlock, Sig, PT, OPCode)$, where the fields previously described in committed transactions are the same, the $ToBlock$ represents the destiny block where this transaction is to be processed. If the $ToBlock$ value is equal to $\emptyset$ and $OPCode$ is a specific value, then the transaction intention is to create a new block with a context or a pure data block. $OPCode$ is an integer that represents a code for the transaction intention, where 1 means the transaction is to create a new pure data block, 2 means the transaction is to create a new block with a context, and 3 means it is a transaction to be appended in a block.

Two functions will be used to summarize the block creation, when $OPCode$ is 1 or 2: Function $NewCBlock$ (Equation 12), which creates a new block with a context starting from a $newS$; and, function $NewPDBlock$ (Equation 14), which creates a new pure data block.

$$\begin{gathered} newCBlock(T, BC) = \\ ((H(p_1(lB(BC))), p_2(p_1(lB(BC))) + 1, \emptyset, NCT_C), \{\}) \end{gathered} \tag{12}$$

$$NCT_C = (P_1(T), p_3(T), p_4(T), VM(\emptyset, P_1(T))) \tag{13}$$

$$\begin{gathered} newPDBlock(T, BC) = ((H(p_1(lB(BC))), \\ p_2(p_1(lB(BC))) + 1,\ PK(p_3(T)), NCT_{PD}), \{\}) \end{gathered} \tag{14}$$

$$NCT_{PD} = (P_1(T), p_3(T), p_4(T), \emptyset) \tag{15}$$

A transaction that has $OPCode$ 3 will be appended to a block. However, the transaction appended is treated differently if the intention is to append a transaction in a pure data block or a block with context. The CT_C function (Equation 16) creates a committed transaction to be appended in a block with a context. On the other hand, function CT_{PD} (Equation 17) will create a transaction to a pure data block.

$$CT_C(T, B) = \begin{cases} (P_1(T), p_3(T), p_4(T), \\ VM(p_4(lastCT(B)))), \\ \text{if } p_4(T) = preCTH(B) \\ \\ B, \text{otherwise} \end{cases} \tag{16}$$

$$CT_{PD}(T, B) = \begin{cases} (P_1(T), p_3(T), p_4(T), \emptyset, P_1(T))) \\ \text{if } p_4(T) = preCTH(B) \\ \\ B, \text{otherwise} \end{cases} \tag{17}$$

$$preCTH(B) = \begin{cases} H(p_1(B)),\ \text{if } |p_2(B)| = 0 \\ \\ H(lastCT(B)), \text{otherwise} \end{cases} \tag{18}$$

$$lastCT(B) = x \,|\, x \in p_2(B) \wedge (\nexists y | y \in p_3(B) \wedge p_3(y) = H(x)) \tag{19}$$

The functions in Equations 16 and 17 are used in function $AppendT$ (Equation 20), which directs a transaction to the correct function type by their $OPCode$ and updates BC. An expiration field was proposed in SpeedyChain [11] to avoid unbalanced blocks, *i.e.*, a block B that has a high number of transactions.

$$AppendT(BC, T) = \begin{cases} (BC - B) \cup CT_C(T, B) \\ \text{if } HasContext(B) = true \\ \\ (BC - B) \cup CT_{PD}(T, B) \\ \text{if } p_3(B) = PK(p_3(T)) \\ \\ BC,\ \text{otherwise} \end{cases} \tag{20}$$

$$B = getBlock(P_2(T)) \tag{21}$$

The algorithm for the main operation for this model is presented in Algorithm 2. In the algorithm, $memPool$ works exactly like presented in Section II. Line 6 checks if the transaction being processed wants to append a new transaction to the blockchain ($OPCode$ 3) and if the destination block exists. If both are true, then the transaction is processed by the consensus algorithm and appended using the $appendT$ function (line 10). When the destination block does not exist, then a transaction creates a new block with a context (line 11). It checks whether the $OPCode$ value is equal to 2 and if the destination block is equal to $emptyset$. If both conditions are true, then a new block will be created (line 15) after being processed by the consensus algorithm. Finally (line 16), the algorithm checks if the transaction wants to create a new pure data block ($OPCode$ 1). If so, then it is checked if there is no other block with the same public key as the signature, then it proceeds to create a new pure data block (line 20).

IV. Implementation

The implementation of the model should be easy to maintain and also easy to incorporate into the existing blockchain technology, in this paper into SpeedyChain. Taking that into consideration, we designed an ideal node architecture (see Figure 2). Three new components are inserted in the SpeedyChain framework: Interface EVM (1), which is an interface

Authorized licensed use limited to: Pontificia Universidade Catolica do Rio Grande do Sul (PUC/RS). Downloaded on August 04,2021 at 13:12:16 UTC from IEEE Xplore. Restrictions apply.

Algorithm 2 Main operation for appendable-block blockchain with context-based model

```
Result: BC //Updated state
Input: memPool, BC //Original state

while(True)
  T = poll(memPool)
  if(exists($p_2$(T)) AND $p_4$(T) = 3)
    ConsensusResponse = performConsensus(T)
    if(consensusResponse)
      broadcast(T)
      BC = appendT(BC, T)
  else if($p_2$(T) = $\emptyset$ AND $p_4$(T) = 2 )
    ConsensusResponse = performConsensus(B)
      if(ConsensusResponse)
        broadcast(B)
        BC = newCBlock(T, BC)
  else if(!exists(PK($p_3$(T))) AND $p_4$(T) =1)
    ConsensusResponse = performConsensus(B)
      if(ConsensusResponse)
        broadcast(B)
        BC = newPDBlock(T, BC)
```

in the SpeedyChain to communicate, through an inter-process communication protocol (2), with an internal EVM (3).

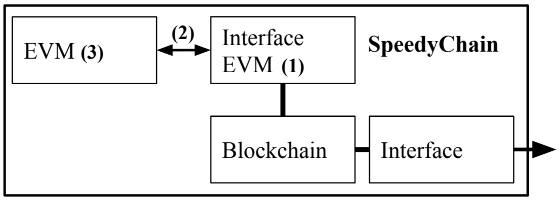

Fig. 2. EVM and SpeedyChain

The process to run a smart contract starts with a new proposed transaction, which contains a bytecode. We assume that all fields of the transaction and the blockchain are correct to receive this generic transaction. As presented in the model, Section III, this transaction targets an existing block. From this block, the last state is extracted and together with the bytecode is passed to the Interface EVM inside the SpeedyChain.

Interface EVM wraps both datas in a JavaScript Object Notation (JSON) format, which is then sent through an inter-process communication channel to the EVM. After sending the JSON, it awaits a return from the EVM with a result. The EVM receives the JSON with the state and bytecode. It changes the virtual machine state to the state received and, using the bytecode as input, processes the request, which yields a resulting bytecode and an updated state. Both are wrapped in a new JSON object and sent as a response to Interface EVM. An error message is sent, if there is any problem when running the smart contract.

The results are unwrapped by the Interface EVM and handed to the blockchain, which will use the updated state to create a committed transaction as described in Section III. If an error is returned, then the transaction is discarded, and no alteration is applied to the blockchain.

The approach shown in Figure 2, allows to use the EVM developed by the Ethereum Foundation. It is implemented in Golang and could not be integrated internally with the SpeedyChain, implemented in Python. The use of that EVM implementation is important for maintainability. Any further update and modification by Ethereum Foundation in the EVM's operations would be inherited in our solution. Note that we are using just the VM from the Ethereum foundation, other modules are not incorporated in our solution. The cryptocurrency, Ether, is intrinsic to the EVM. However it is not used, since there are no rewards for blocks creation and no way to create Ether in our implementation. Generally in Ethereum, the cost of computation, named gas, is paid to miners using Ether. However, this feature was removed in our solution. We adopted an unlimited gas limit (computation power).

V. Evaluation

In order to evaluate context-based smart contracts in appendable-block blockchains, we applied our solution in four testing scenarios (see Table I). Every scenario was performed on a Virtual Machine (VM) with 4-core processor, 16GB of memory and 64MB of graphics memory running Ubuntu 18.04 operating system using a Virtual Box hypervisor over a Macbook Pro with 2.3 GHz 8-Core Intel Core i9 processor, 32GB DDR4 memory. In order to create a container-based network to emulate network equipment, gateways and devices, the Core Emulator [31] was used. For all scenarios, a network with ten gateways was simulated (see Figure 3). In all scenarios, a GPS tracking smart contract was used, which uses an approximation to calculate the distance between GPS positions. The source in Solidity language is available on Github[1].

TABLE I
Evaluated scenarios

Scenario	Description
A	Sequential execution of 1,000 smart contracts without external load
B	Sequential execution of 1,000 smart contracts with an additional load of 50 devices per gateway and 100 transactions per device, *i.e.*, 5,000 transactions per gateway and 50,000 transactions in total
C	10 parallel context with 100 smart contracts transactions per context (1,000 in total) without external load
D	10 parallel context with 100 smart contracts transactions per context (1,000 in total) with an additional load of 50 devices per gateway and 100 transactions per device, *i.e.*, 5,000 transactions per gateway and 50,000 transactions in total

Scenario A was performed to establish a baseline for smart contracts execution in appendable-block blockchains. One hundred smart contracts operations were executed to compute the distance between a device and a target using GPS information. We simulated ten devices with sequential computation of smart contracts operations (total of 1,000 sequential transactions) in the same gateway. Similarly to *Scenario A*,

[1] https://github.com/conseg/GPSTracker

Authorized licensed use limited to: Pontificia Universidade Catolica do Rio Grande do Sul (PUC/RS). Downloaded on August 04,2021 at 13:12:16 UTC from IEEE Xplore. Restrictions apply.

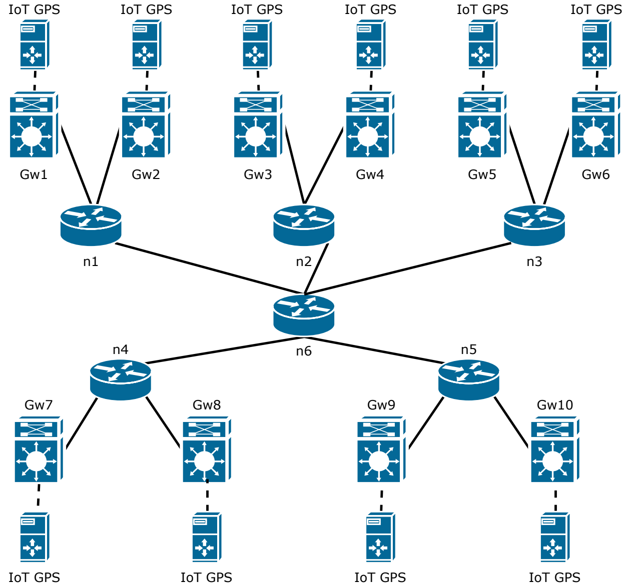

Fig. 3. Emulated gateways architecture

Scenario B performed the same number of sequential smart contracts computations, however, with an extra load of normal type of transactions in every gateway. A normal transaction in this experiment is a random input of data in a pure data block. The load was simulated through the insertion of 50 devices for each gateway and 100 "normal" transactions produced by each device. In total, 50,000 additional "normal" transactions were simulated. These transactions were produced by 500 devices.

Different from previous described scenarios, *Scenario C* used the proposed context-based smart contracts in appendable-block blockchains. To do that, this scenario considered ten devices connected to ten different gateways and requesting 100 smart contracts operations (1,000 operations, the same computations as in *A* and *B*) in parallel, without any extra load in the gateways. Similarly to *C*, *Scenario D* adopted context-based smart contracts, with the same number of transactions operations as C, but with the same extra load present in *B*.

A. Metrics

In order to perform an evaluation of the proposed context-based smart contracts in SpeedyChain, metrics called T1, T2, T3, T4, T5, and T6 were used:

- **T1**: Time to perform consensus, insert a new block (first time that device is connected) and replicate it to all gateways;
- **T2**: Time to add and replicate a device block to all gateways (after consensus);
- **T3**: Time to perform consensus, insert a new special block for smart contracts and update EVM with the new smart contract bytecode;
- **T4**: Time to insert a transaction into the blockchain;
- **T5**: Time to run a smart contract in the EVM and update the blockchain;
- **T6**: Time to run all smart contracts evaluated (1,000 contracts operations).

All metrics are represented by an average time of ten repetitions for each scenario. A confidence level of 95% was achieved.

B. Results

Context-based smart contracts execution can impact the consensus algorithms (**T1**) as presented in Table II. We can see that time to perform consensus can increase ≈ 65% when using context-based smart contracts (*Scenario C*) when compared to a sequential smart contracts execution (*Scenario A*) in a scenario without an extra load in gateways. When comparing the scenario with extra load ("with normal transactions"), we can observe that consensus time is increased by less than 13% when comparing with (*Scenario D*) and without (*Scenario B*) the context-based approach. This can be explained by the time required by the gateways to process smart contracts, affecting the time to perform consensus.

Also, we can observe similar behavior in the time to perform consensus and update all the gateways to ensure a global view of the blockchain (**T2**) presented in Table II. In this case, we can observe a higher increment comparing *Scenarios C* to *A* (≈ 85%), and *D* to *B* (≈ 14%). This result shows that it also affects not only the leader of the consensus but also all the gateways.

Table II presents the time to insert special blocks for a smart contract (**T3**). In all scenarios, the time required was higher than the average presented in **T2**. That can be explained by the communication and processing performed in the EVM. Additionally, a lower difference was observed when comparing *D* to *B* (≈ 11%), than comparing *C* to *A* (≈ 81%).

Time to insert transactions into the blockchain (**T4**) was less affected than block insertion by the proposed context-based smart contracts solution, as presented in Table II. *Scenario C* increased ≈ 23% over *A*, and *D* increased ≈ 7.5% over *B*. Comparing the usage of context-based smart contracts adoption in **T3** and **T4**, we can observe that the impact in mean time to insert transactions (**T4**) was lower than in block insertion (**T3**).

One important measure is how much the processing of smart contracts is affected by individual executions (**T5**). We can observe in Table II that *Scenario C* increased in ≈ 18% over *A* and *Scenario D* increased in ≈ 23% over *B*. This shows that the parallel approach proposed in context-based smart contracts affect individual insertions. It can be explained by the larger number of messages exchanged by nodes. Although, 79.58ms (in the worst case) to receive the result of computation still very good considering the usual GPS update rate of one update per second (1Hz) [32].

Finally, as presented in Figure 4, time to perform calls for every smart contract (**T6**) was reduced when using context-based smart contracts in parallel execution (*Scenarios C* and *D*) compared to "traditional" sequential execution (*Scenarios A* and *B*). Differently to the other metrics, **T6** is presented in seconds for each scenario. *Scenarios A* and B required

Authorized licensed use limited to: Pontificia Universidade Catolica do Rio Grande do Sul (PUC/RS). Downloaded on August 04,2021 at 13:12:16 UTC from IEEE Xplore. Restrictions apply.

TABLE II
RESULTS SUMMARY

Scenario	**T1** (in milliseconds)	**T2** (in milliseconds)	**T3** (in milliseconds)	**T4** (in milliseconds)	**T5** (in milliseconds)	**T6** (in seconds)
A	19.54 ± 0.24	$46.53{\pm}0.48$	61.24 ± 0.65	0.97 ± 0.003	29.17 ± 0.03	31.22 ± 0.43
B	90.98 ± 0.77	279.64 ± 2.53	305.22 ± 33.45	2.81 ± 0.008	64.18 ± 0.40	70.40 ± 0.97
C	32.40 ± 1.43	85.59 ± 3.55	111.03 ± 4.45	1.20 ± 0.10	34.47 ± 0.10	3.46 ± 0.12
D	102.67 ± 0.87	320.24 ± 2.93	340.03 ± 40.04	3.02 ± 0.01	79.58 ± 0.72	6.96 ± 0.48

an average of $31.22s{\pm}0.43s$ and $70.40s{\pm}0.97s$, respectively. Although, *Scenarios C* and *D* required only $3.46s{\pm}0.12s$ and $6.96s{\pm}0.48s$. While context-based approach (*C* and *D*) presented an impact in the main operations (**T1**, **T2**, **T3**, **T4** and **T5**) of the blockchain, **T6** required around 10% of the time to perform all smart contracts than in the sequential approach (scenarios *A* and *B*).

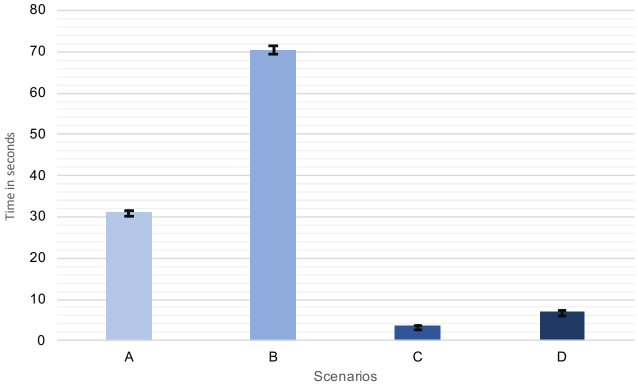

Fig. 4. **T6**: Time to execute 100 calls for each 10 different smart contracts

C. Discussion and threats to validity

Results presented in the Evaluation section show the possible gains when using context-based smart contracts. Reducing the total time of smart contracts can contribute to a more distributed execution of smart contracts than the current solutions. The adoption of appendable-block blockchain allowed a parallel execution, resulting in smaller time to perform smart contracts operations (as presented in **T6**).

Nonetheless, overhead is introduced by context-based smart contracts, especially when comparing a scenario without an extra load on gateways. Context-based approach increased in more than 60% the time to achieve consensus for new blocks in the blockchain (**T1**, **T2** and **T3**) in scenarios without extra load. However, scenarios with extra load (simulating a real blockchain scenario) presented a small increase for the same indicators (less than 15%). Also, it is important to note that the block insertion is performed only once for each context.

There are some threats to the validity on the results presented in Section V-B. The first threat is related to hardware capability. In this work, we did not use physical IoT devices and GPS. However, we used the same cryptography algorithms and methods than those that were adopted by Lunardi *et al.* [7] in their experiments (using real hardware). Consequently, devices using IoT hardware should be capable of executing the same operations, but probably with different performance.

Additionally, the evaluation did not consider possible signal and communication problems. Consequently, the experiments did not take into account issues that mobile devices and/or gateways can produce. Hence, this threat should be further discussed and mitigated in a future work.

Finally, threats related to security issues that the proposed approach can introduce. For example, the impact of tampered devices producing invalid data (*e.g.*, invalid coordinates) were not considered. This specific threat should also be better addressed in future work.

VI. FINAL CONSIDERATIONS & FUTURE WORK

In this paper, we presented a new model for context-based smart contracts that can be applied to appendable-block blockchains. This model expands the possibilities that the appendable-block blockchains can be used for. Also, it can allow improvements in the performance of smart contracts computation through parallel execution.

Results presented in the evaluation indicate that the execution of smart contracts can be reduced when they are classified in independent contexts. For example, in the scenario with additional load, the time to perform all the smart contracts computations was reduced from more than 70 seconds to less than 7 seconds. However, the proposed context-based smart contracts increased the time to perform consensus, due to the usage of gateways' processors to compute smart contracts, although, block insertion is performed only once per context. Additionally, in the scenario with extra load, this increase was reduced when compared to a scenario without extra load.

Due to space limitation, in this paper we did not discuss security issues that could be introduced by this new model. For example, there is no protection against replay attacks, an attack where a transaction is copied by a malicious user and sent to be processed again.

As future work, we intend to evaluate the proposed approach using actual hardware, in special to evaluate the impact that both latency and constrained hardware can have in context-based smart contracts. Also, we expect to reduce the overhead introduced by the context-based approach. Additionally, we intend to evaluate possible security issues and possible new attacks that can be explored in this approach. Finally, we intend to expand the context-based smart contracts model to other blockchains, such as Ethereum and Hyperledger Fabric.

Authorized licensed use limited to: Pontificia Universidade Catolica do Rio Grande do Sul (PUC/RS). Downloaded on August 04,2021 at 13:12:16 UTC from IEEE Xplore. Restrictions apply.

References

[1] A. P. Sheth and J. A. Larson, "Federated database systems for managing distributed, heterogeneous, and autonomous databases," *ACM Computer Survey*, vol. 22, no. 3, pp. 183–236, September 1990.

[2] J. A. Berkowsky and T. Hayajneh, "Security issues with certificate authorities," in *2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON)*, October 2017, pp. 449–455.

[3] V. Dedeoglu, R. Jurdak, A. Dorri, R. C. Lunardi, R. A. Michelin, A. F. Zorzo, and S. S. Kanhere, *Blockchain Technologies for IoT*. Springer Singapore, 2020, pp. 55–89.

[4] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," in *2017 IEEE International Congress on Big Data (BigData Congress)*, June 2017, pp. 557–564.

[5] S. R. Niya, D. Dordevic, A. G. Nabi, T. Mann, and B. Stiller, "A platform-independent, generic-purpose, and blockchain-based supply chain tracking," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, May 2019, pp. 11–12.

[6] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the internet of things," *IEEE Access*, vol. 4, pp. 2292–2303, May 2016.

[7] R. C. Lunardi, R. A. Michelin, C. V. Neu, and A. F. Zorzo, "Distributed access control on IoT ledger-based architecture," in *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, April 2018, pp. 1–7.

[8] R. C. Lunardi, H. C. Nunes, V. S. Branco, B. H. Lippert, C. V. Neu, and A. F. Zorzo, "Performance and cost evaluation of smart contracts in collaborative health care environments," in *14th International Conference for Internet Technology and Secured Transactions (ICITST)*, 2019, pp. 1–10.

[9] M. Singh and S. Kim, "Trust bit: Reward-based intelligent vehicle commination using blockchain paper," in *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, Feb 2018, pp. 62–67.

[10] S. Kushch and F. Prieto-Castrillo, "Blockchain for dynamic nodes in a smart city," in *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, April 2019, pp. 29–34.

[11] R. A. Michelin, A. Dorri, M. Steger, R. C. Lunardi, S. S. Kanhere, R. Jurdak, and A. F. Zorzo, "Speedychain: A framework for decoupling data from blockchain for smart cities," in *15th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous)*, November 2018, pp. 145–154.

[12] M. Merlini, N. Veira, R. Berryhill, and A. Veneris, "On public decentralized ledger oracles via a paired-question protocol," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, May 2019, pp. 337–344.

[13] 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)*, April 2017, pp. 253–256.

[14] L. M. Bach, B. Mihaljevic, and M. Zagar, "Comparative analysis of blockchain consensus algorithms," in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, May 2018, pp. 1545–1550.

[15] R. C. Lunardi, R. A. Michelin, C. V. Neu, H. C. Nunes, A. F. Zorzo, and S. S. Kanhere, "Impact of consensus on appendable-block blockchain for IoT," in *Proceedings of the 16th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous)*, 2019, p. 228–237.

[16] D. Lee, "Hash function vulnerability index and hash chain attacks," in *3rd IEEE Workshop on Secure Network Protocols*, Oct 2007, pp. 1–6.

[17] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ecdsa)," *International Journal of Information Security*, vol. 1, no. 1, pp. 36–63, August 2001.

[18] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[19] A. Malatras, "State-of-the-art survey on p2p overlay networks in pervasive computing environments," *Journal of Network and Computer Applications*, vol. 55, pp. 1 – 23, May 2015.

[20] A. F. Zorzo, H. C. Nunes, R. C. Lunardi, R. A. Michelin, and S. S. Kanhere, "Dependable IoT using blockchain-based technology," in *2018 Eighth Latin-American Symposium on Dependable Computing (LADC)*, October 2018, pp. 1–9.

[21] Peercoin Foundation, "Peercoin documentation," February 2019. [Online]. Available: https://docs.peercoin.net/

[22] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI)*, February 1999, pp. 173–186.

[23] M. S. Ali, M. Vecchio, M. Pincheira, K. Dolui, F. Antonelli, and M. H. Rehmani, "Applications of blockchains in the internet of things: A comprehensive survey," *IEEE Communications Surveys Tutorials*, vol. 21, no. 2, pp. 1676–1717, Secondquarter 2019.

[24] Ethereum Foundation, "Ethereum documentation," February 2019. [Online]. Available: http://ethdocs.org/en/latest/index.html

[25] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, and et al., "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference (EuroSys)*, April 2018, pp. 1–15.

[26] D. M and N. B. Biradar, "IOTA-Next Generation Block chain," *International Journal of Engineering and Computer Science*, vol. 7, no. 04, pp. 23 823–23 826, April 2018.

[27] Ethereum Foundation, "Ethereum white paper," Dec. 2018. [Online]. Available: https://github.com/ethereum/wiki/wiki/White-Paper

[28] A. Aldweesh, M. Alharby, M. Mehrnezhad, and A. Van Moorsel, "Opbench: A CPU performance benchmark for ethereum smart contract operation code," in *2019 IEEE International Conference on Blockchain (Blockchain)*, July 2019, pp. 274–281.

[29] A. Aldweesh, M. Alharby, E. Solaiman, and A. van Moorsel, "Performance benchmarking of smart contracts to assess miner incentives in ethereum," in *2018 14th European Dependable Computing Conference (EDCC)*, September 2018, pp. 144–149.

[30] Linux Foundation, "Hyperledger blockchain performance metrics," December 2018. [Online]. Available: https://www.hyperledger.org/resources/publications/blockchain-performance-metrics

[31] J. Ahrenholz, C. Danilov, T. R. Henderson, and J. H. Kim, "Core: A real-time network emulator," in *27th IEEE Military Communications Conference (MILCOM)*, November 2008, pp. 1–7.

[32] J. Y. Huang, C. H. Tsai, and S. T. Huang, "The next generation of GPS navigation systems," *Communications of the ACM*, vol. 55, no. 3, pp. 84–93, March 2012.

Authorized licensed use limited to: Pontificia Universidade Catolica do Rio Grande do Sul (PUC/RS). Downloaded on August 04,2021 at 13:12:16 UTC from IEEE Xplore. Restrictions apply.