A Number of Conceptual Scalable Peer-to-Peer Message-Routing and Node-Organizing Multi-Tiered Blockchain Architectures for IoT

1st 2nd

***Abstract*—Blockchain's inherent characteristics, namely its immutability, security, and decentralization, have garnered significant attention in non-monetary applications, particularly in the context of the Internet of Things (IoT). However, the substantial volume of transactions generated by IoT devices at a high rate presents a challenge, since current blockchain models lack the required scalability, throughput, and consensus efficiency essential to effectively support them. Therefore, we propose and compare several conceptual multi-tiered blockchain node-organizing architecture designs. The proposed designs are constructed based on various algorithmic concepts, although they functionally achieve consensus using the same mechanism that we propose. The objective is to enhance blockchain scalability and efficiency by constraining the number of hops and nodes involved. Our proposed architectures are: i) Self-Scalable Dynamic (SSD) architecture, ii) Partitioned architecture, iii) Ring of Rings architecture and iv) Improved Ring of Rings architecture. At the heart of the proposed architectures lie mechanisms for node lookup and message routing to easily locate nodes and route messages, along with a consensus mechanism adaptable to each proposed design. Based on our proposed designs, transaction and block verifications as well as block creation are performed by a logarithmically bounded number of nodes, compared to the number of nodes at the levels below. Consensus time and scalability are significantly improved with the increase of the number of levels. Simulations of the SSD architecture and algorithm in particular, as an illustrative example of a proposed multi-tiered design, offer a robust proof of concept with promising results that outperform both flat and two-tiered models. Our proposed architecture designs are application and algorithm-agnostic, and a specific design may be preferred depending on the distinct requirements of each application.**

***Index Terms*—Blockchain, IoT, Consensus, Scalability, Load balancing, Decentralization, Security, Management Overhead, Look-up, Multi-Tiered, Node Look-up, Message Routing.**

I. Introduction

IoT devices (IoTDs), characterized by their limited resources, heterogeneity and security issues may potentially benefit from blockchain's salient features such as decentralization, security and immutability. However blockchain's high bandwidth overhead, latency and limited throughput defeat its intended purpose of supporting IoTDs [1], [2]. Since a typical blockchain (BC) is a hypothetical peer-to-peer overlay design over a real, physical network, where uncontrollable delays take place, consensus time can be calculated by counting the number of hops required for a transaction (Tx) to reach BC within the overlay network. The purpose of this study is to present and compare a variety of algorithmic concepts for building efficient BC architecture designs that reduce the number of hops required for consensus and assess latency improvements in the overlay network, presuming that the physical network will eventually exhibit the gains. The designs may serve as general-purpose scalable peer-to-peer node-lookup and message-routing solutions for distributed systems.

Considering current BC node-organizing models, we can classify them as: 1- The flat model where all nodes are located at the same level and may generate transactions and build blocks. This model suffers from scalability limitations, high traffic overhead and delays, since all nodes must be visited to achieve consensus. 2- The two-tiered model where block-building IoTDs (referred to as *Cluster Heads (CHs)*) located on the top level manage the clustered Tx-generating IoTDs (simply referred to as *IoTDs*), located at a lower level. This model suffers from the same drawbacks as the flat model, only reflected at the top level.

Our proposed designs aim to address current BC flat and two-tiered models' scalability and delay issues by limiting the visited nodes to only the nodes required for achieving consensus and by expanding two-tiered models into multi-tiered architectures (forests) where all nodes are clustered and organized according to their capabilities throughout the heirarchy. All nodes are IoTDs s.t. a Tx may only be generated by an IoTD located at the bottom level (level 0), simply referred to as *IoTD*, while Tx and block verification as well as block creation are performed by the topmost IoTDs (at level N), referred to as *CHs*. The rest of the nodes, located at levels [1..N-1], are simply referred to as *CHs* and they help maintain the multi-tiered structure. A Tx is generated by two IoTDs (Tx parties) at the bottom of the structure and propagates upwards towards the topmost level to be verified and appended into a block, as in Fig. 1. Once a topmost CH receives a predefined number of verified transactions, it may build a block. Blocks are built and verified at the topmost level.

Each level has a logarithmically bounded number of nodes less than the number of nodes at the levels beneath it. Hence, the deeper the forest, the fewest the topmost CHs that verify transactions and blocks, the less the consensus delays, and the more scalable the model becomes. Each cluster is exactly managed by one CH that holds its managed tree and/or cluster nodes' information. Clusters' size are comparable by level to

ensure a comparable consensus time for all IoTDs. Each node is labeled with a Cluster Head Address Chain (CHAC) to determine its logical position in the structure as well as its chain of parents', s.t. CHAC length of a node is equal to the node's depth+1. CHAC enables the upward propagation of Tx from IoTDs towards topmost CHs to occur in $O(1)$ no matter how deep the forest is, which further reduces the consensus delays.

Searching a node can be performed with a reliable and efficient (probabilistically bounded) delay at the price of some complexity of the initial construction and overall maintenance of the designs.

Although constructed based on diverse design concepts, our proposed architectures achieve consensus through the same mechanism just described, resulting in substantial improvements over flat and two-tiered models. Nevertheless, the asymptotic analysis of the consensus mechanism, as it adapts to each design, varies. Despite the application and algorithm-agnostic nature of our designs, the suitability of a particular design depends on the unique requirements of each application.

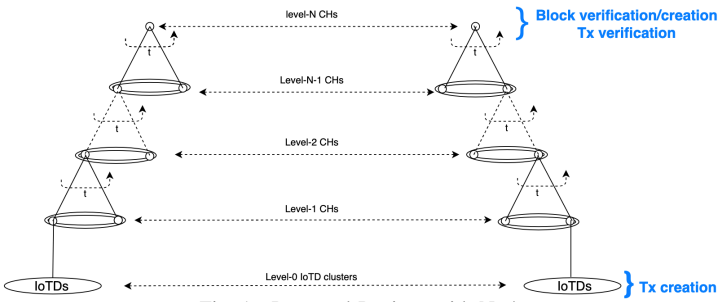

Fig. 1. Proposed Designs with N=4.

Our contributions are: i) Self-Scalable Dynamic (SSD) architecture, ii) Partitioned architecture, iii) Ring of Rings architecture and iv) Improved Ring of Rings architecture.

The rest of the paper is organized as follows. Section II reviews existing BC node-organizing designs aiming to reduce consensus delays. Section III outlines the proposed architecture designs. Section IV introduces a comparison among the proposed designs. Section V summarizes our simulation results and section VI concludes the paper.

II. Literature Review

In this section, we review designed architectures that are relevant to our multi-tiered models. The fundamental scalability and efficiency issues are some of the main obstacles to overcome when implementing BC into IoT ecosystems. Scalability is evaluated based on different criteria including the number of supported IoTDs as well as Tx throughput. An example two tiered distributed model that addresses scalability is Lightweight Scalable BC (LSB) [3]. In LSB, IoTDs are clustered and consensus is simplified by putting in place a trust system that allows some Tx to skip verifications, hence, improving consensus time. Another implementation for a two-tiered architecture where Proof of Work (PoW) [4] consensus is modeled, is proposed by [5]. An implementation of an existing BC platform naturally characterized by a scaled throughput and low latency is introduced in [6] as an architecture built on Ethereum [7], [8], leveraging its smart contracts [9], [10].

In the first couple of architectures, the two-tiered designs are rigid s.t. there is no possibility to add more levels and there is a strict distinction between IoTDs and CHs. Consequently, they don't exchange positions or roles, hindering scalability to meet the demands of today's IoT. In these designs, the structure is pre-built before the consensus mechanism is initiated, which may not always align with the requirements of a dynamically operating BC that must continually expand on the fly. Furthermore, the designs lack a node labeling mechanism to easily locate nodes within the structure. Additionally, they overlook the potential improvement in consensus when two IoTDs participating in the same Tx co-exist within the same hierarchy tree (referred to as *Colocality*). Recognizing this synergy could lead to a substantial enhancement in consensus, a crucial benefit associated with the necessity to establish a hierarchy in the first place.

The existing BC designs (some of which discussed above) suffer from: i) limited scalability, ii) low throughput, iii) consensus inefficiency, iv) a lack of a mechanism to logically locate nodes, and v) a lack of node colocality consideration.

III. The Architectures' Details

Our proposed design options are constructed based on distinct conceptual designs, which reflects on the processes of the CHAC formulation, the asymptotic analysis of the consensus mechanism and the timing of the architecture building phase compared to BC functionality. The designs are as follows:

A. Option 1 - SSD Architecture and SSD Algorithm

Design option 1 is an elastic self-scaling dynamic (SSD) architecture that aims to maintain a constant equipoise between node management overhead and consensus decentralization. All nodes are clustered except the topmost level and the architecture is dynamically built and modified while BC functionality is performed. This approach aims to enhance scalability and flexibility in constructing the model, as it allows IoTDs and CHs to dynamically exchange positions and roles within the structure. Additionally, new architecture levels can be added or existing levels can be removed as needed.

A forest of trees is built bottom-up by performing cascading splits and merges similarly to B-trees [11] s.t. a level can be eliminated or added at the top of all levels. While nodes join and/or leave the system, cluster sizes may get too small to achieve decentralization or too large to be managed according to parent CHs' capacity, thus thresholds are set (MUT stands for Management Upper Threshold and CLT stands for Control Lower Threshold). A cluster that becomes too small (smaller than CLT) merges with the smallest cluster on the same level, within the same subtree, to make a bigger cluster; while a cluster that becomes too large (larger than MUT), splits into two clusters. IoTDs and CHs may then exchange roles and positions within the structure.

SSD algorithm helps load balance node management overhead versus consensus decentralization as it continuously triggers cluster splits and merges according to the thresholds while IoTDs dynamically join and leave the system. To maintain the structure integrity (i.e. each cluster is managed by exactly one CH), every split is accompanied by a merge and vice versa. Due to the cascading splits and/or merges, the architecture autonomously, de-centrally and dynamically readjusts as it expands and/or contracts. SSD algorithm also verifies the size of a resulting cluster after a merge, in case a further split is required, and verifies the size of each resulting cluster after a split, in case a further merge is required, which makes the SSD algorithm cyclic. SSD algorithm is application agnostic and converges according to each application's needs.

The construction algorithm is as follows:

1) Starting at level 0 (the bottom level of the hierarchy), all IoTDs are labeled using sequential numbers, each with a one-digit CHAC.
2) All IoTDs are clustered using comparable cluster sizes, setting the MUT and CLT thresholds for level-0 clusters s.t. a cluster's size is not less than CLT.
3) More powerful IoTDs, one per cluster, are promoted as level-1 CHs (at the level right above IoTDs) s.t. each CH manages one cluster. A CH is labeled with a one-digit CHAC according to a different naming strategy to differentiate between CHs and IoTDs. IoTDs managed by a CH copy their CH's CHAC and append it to their own CHAC. MUT and CLT are set for level 1, the current unclustered topmost level.
4) Upper levels are created recursively, by adding IoTDs to each level-0 cluster, overflowing the clusters s.t. a cluster size becomes larger than the per-level MUT. The SSD algorithm then automatically detects MUT violation and splits each level-0 cluster into two, not less than CLT comparable size clusters (while maintaining the structure's balance by mimicking splits for the level's clusters). One CH per newly created cluster is promoted to join the level above, and an extra digit is appended to its CHAC. The promoted CH's CHAC is then communicated and appended to the CHAC of all nodes managed by the CH. A new level is created by overflowing the clusters of the current topmost level and promoting one CH per topmost cluster to form a new unclustered topmost level. The new level becomes the current topmost level and remains unclustered and the level below it (the previous topmost level) is clustered while new MUT and CLT are set for the new level. Consecutive MUT violations with cascading splits and CH upward promotions take place while adding new levels.

For clustering, the K-means Greedy [12]–[15] algorithm is recommended due to its linear complexity, uniform modularity[1] [16], [17] and compatibility with our SSD algorithm. The SSD algorithm takes a CLT input value that's equal to k, representing the number of K-means Greedy clusters to be built, making it a seamless plug-and-play integration.

The cost of building the architecture depends on the clustering algorithm used. In case K-means Greedy algorithm is applied, all IoTDs are clustered to form level-0 clusters, and subsequently, all levels are clustered except for the current topmost level. Consequently, the construction cost linearly increases with the number of CHs on each added level, as long as it is not the current topmost level. Nevertheless, SSD maintenance cost is high as the system must maintain a number of data structures to store the node-cluster-CH relationships. Consequently, altering the position of a single node results in a substantial number of operations, mirroring findings from a comparable research [18].

Two IoTDs (Tx parties) may participate in one Tx s.t. Tx verification takes place at a topmost parent node where both parties' public key information are present. A Tx propagates from an IoTD Tx party upwards towards topmost CHs in $O(1)$, thanks to the use of CHAC. When both Tx parties are located in the same tree (referred to as *Colocality*), Tx verification takes place at the tree root in common for both parties. Otherwise, when Tx parties are non-colocal, after a Tx propagates from one IoTD towards its topmost parent, all topmost CHs are visited and searched for the other party information, hence Tx verification is in $O(maxprop)$ (maximum propagation delay towards topmost nodes). Also block verification by all topmost CHs takes $O(maxprop)$. Thus consensus takes $O(1)$ plus twice $O(maxprop)$, worst case, when Tx parties are non-colocal; otherwise it takes $O(1)$ plus $O(maxprop)$, best case. The more we increase the number of architecture levels, the more we improve node colocality, hence consensus efficiency and the overall scalability, as shown in Fig. 2.

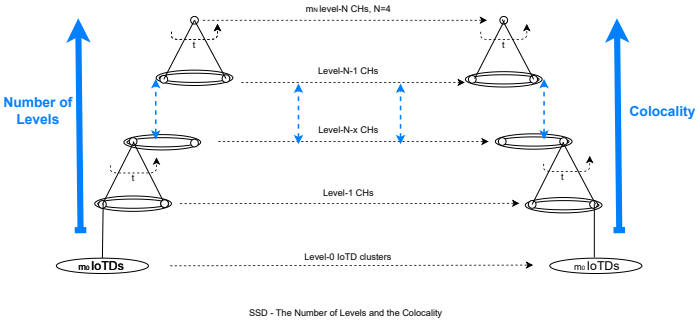

Fig. 2. SSD - The Number of Levels and the Colocality.

B. Option 2 - Partitioned Architecture

Design option 2 consists of a multi-tiered forest architecture where IoTDs are partitioned into sections to form a

[1]The modularity function is biased when the algorithm identifies the clusters with lower number of members to join a neighbor cluster having the highest number of members. This produces crowded clusters with a CH overburdened with management overhead, which is not a desirable feature.

hypothetical ring. Each section exploits a Distributed Hash Table (DHT)-like [19] approach. The construction of the forest occurs top-down, by applying modulo reduction to the hash of the IoTDs' physical (i.e., MAC) addresses, with subsequent levels being added, as in Fig. 3. In a recursive manner, IoTDs are logically partitioned and organized into sections, positioned clockwise. Simultaneously, labeled CHs are assigned to each section to construct a forest. The forest is abstracted into a ring-like graph where IoTD and CH roles are clearly differentiated. This design is rigid: once constructed, roles and positions of IoTDs and CHs can not be exchanged, and the structure cannot be altered by adding or removing levels. The construction of this multi-tiered architecture takes place prior to the initiation of BC functionality (Tx generation and consensus processes).

CHORD [20], a well-known Distributed Hash Table lookup system, maps node identifiers and keys onto the same virtual ring. CHORD is based on utilizing a hash function that generates m-bit unique identifiers for both nodes and keys. Locating a node on a CHORD ring takes $log\, M$, where M is the number of CHORD nodes.

To build the architecture, all nodes' MAC addresses are hashed in a ring. All nodes are recursively partitioned using a modulo reduction based on the branching factor in order to build the CHAC of all levels in a top-down approach. Once level-1 CH (the level right above IoTDs) is reached, CHORD is built on each IoT cluster (subsection), so that IoTDs can easily join and leave the structure while all upper-level CHs are fixed (and thus, do not exchange roles and positions with IoTD nodes). The resulting structure is an abstraction of a forest in a ring-like structure.

The construction algorithm is as follows:

1) A hash function (a base consistent hash function such as SHA-1) is applied to distribute all IoT MAC addresses evenly to form a hypothetical logical ring. In this ring, the nodes can be relocated, so the hashed addresses are partitioned into equal sections using a modulo reduction of their hash values with a modulus that is a power of two to form power of 2 sections of the ring (e.g. $\mod 16 = 2^4$ to form 16 distinct sections). Each section may be identified by a label (one CHAC digit) and each section is considered hypothetically separate from other sections on the ring. For each section, one more powerful CH is assigned. Flagged as "CH", each CH is logically located at the counterclockwise border of each section on the ring. The CHs represent the successors of each section (mimicking the CHORD successor concept, though CHORD is not to be applied yet). The CHs also play the role of level-N (topmost) CHs which are the forest roots. All nodes and CHs are labeled with a one-digit CHAC that is equal to the hash result.
2) The hash of the IoTDs' MAC address excluding the least significant b bits (here $b = 4$, for a branching factor=16) that represent the modulo of the previous hash, is saved. The resulting hash is then considered as the current level's hash (level-$N-1$ hash), which is equal to the current level's CHAC digit. Then the hash is used to further partition each of the node sections resulting from the previous step, again using a modulo reduction function of the branching factor. Each of the previous sections is thus divided into 16 smaller sections and all the nodes are re-partitioned accordingly. Then a CH (successor) is assigned to each resulting section. The CHs as well as the new sections' nodes are labeled by appending a new one-digit CHAC to the previous CHAC, representing the modulo value of the level, s.t. the resulting CHAC of any node at depth i represents the one-digit CHAC resulting from the current level's modulo/partitioning round concatenated to the CHAC digits resulting from the previous rounds.
3) The previous step is repeated, so with every iteration, the following is performed to build the forest top-down:
    a) A new level hash that is b bits less than the hash of the previous iteration, is saved.
    b) Each of the previous sections is partitioned into new sections according to the branching factor.
    c) CHs are assigned to each new section as successors. The CHs are labeled with a one-digit CHAC that is appended to the old CHAC from the previous iteration. The new sections' nodes are also labeled with the same CHAC as the new CHs, s.t. CHAC length increases with every level (with every partitioning/CH assignment round).

    The iterations continue using the same modulo reduction based on the branching factor, for every level until the desired number of levels is reached for the structure. We end up with a number of sections containing IoTDs followed by successors that are labeled with the same CHAC as that of the IoTDs. Each section is identified according to the CHAC label of its nodes. The section's successor nodes are identified as the distinct flagged nodes located at the counterclockwise border of each section on the ring, and they represent the CHs throughout the forest levels. The rest of the section's nodes (labeled similarly as the sections' successors) are the forest leaves (IoTDs) located at level-0 of the structure. At the end of this step, all levels are represented, each with an $i+1$-digit CHAC (logical address), where i is the depth of the level and where each CHAC digit position represents the forest level it belongs to.
4) CHORD is applied on each IoTD cluster (IoTDs labeled with the same CHAC). CHORD necessitates the generation of m-bit unique identifiers for both CHORD nodes and keys. Consequently, to identify and locate each IoTD within a cluster, CHORD's node ID consists of the most significant m bits taken from the hash of each IoTD IP address, while CHORD's key ID consists of the m bits taken from the rest of the bits resulting from the modulo hash reduction of the node's MAC address excluding the least significant bits used so far to create the cluster's CHAC. Hence, CHORD relates

IoTDs' physical addresses to their logical addresses. Choosing a small number of IoTDs within one cluster as CHORD key successor nodes, then mapping the rest of the cluster IoTDs to the successors according to CHORD takes $O(log\,X)$ where X is the IoTD cluster size.

The cost of building the architecture is measured once for all, since the structure cannot be modified once built (hence, it has no maintenance cost). With every level to be added, we perform a hashing round for all nodes. Consequently, the construction cost increases linearly with the number of levels to be built. In addition, once all levels are constructed, the cost of constructing a CHORD ring for each resulting IoTD cluster, must be considered.

To achieve consensus, a Tx needs to propagate from an IoTD Tx party towards the other party in order to obtain the latter's public key. Knowing the MAC address of the other party makes it easy to determine its CH's CHAC by performing a number of modulo-b reductions on the party's MAC address hash, that number is equal to the number of the structure's levels. Reaching the other party's CH takes $O(1)$ (thanks to the use of CHAC), while looking for the other party within its cluster using CHORD takes $log\,X$, where X is the cluster size. Then the Tx propagates to the topmost level in $O(1)$ (using CHAC) in order to be verified by all topmost CHs in $O(maxprop)$. Also a block is verified by all topmost CHs in $O(maxprop)$. Hence, consensus takes $O(number\,of\,structure's\,levels)$ plus $O(2)$ plus $O(log\,X)$ plus twice $O(maxprop)$, in case both IoTDs are located in different clusters (assuming the sizes of IoTD clusters are comparable throughout the structure), worst case. Tx propagation from an IoTD Tx party towards another Tx party located within the same cluster is in $O(log\,X)$ (using CHORD), otherwise. Hence, considering that $O(number\,of\,structure's\,levels)$ plus $O(2)$ are equal to a constant number, then consensus complexity analysis is almost equal for both worse and best cases.

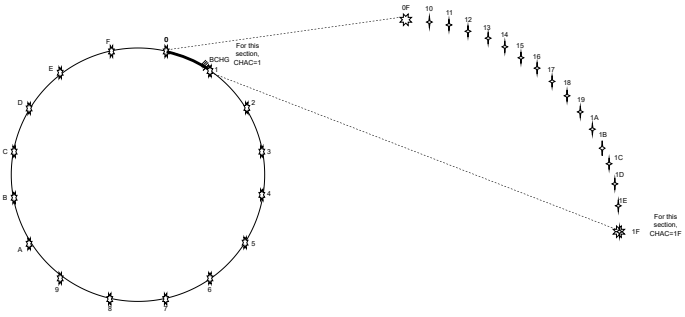

Fig. 3. Partitioned Architecture.

C. Option 3 - Ring of Rings

Design option 3 is an architecture that permits flexible assignment of CHs and provides the flexibility to add or remove levels. It is based on mapping the node's location within the forest to a corresponding location within DHT-like tables.

The construction algorithm is based on the idea of building a DHT at the CH of each IoTD cluster. Next, some of the IoTDs are promoted as CHs for subsequent levels, gradually and recursively. The final result is a representation of the forest structure, abstracted in a ring-like graph.

We construct the architecture bottom-up, drawing on various node identification and lookup approaches proposed by [20]–[24], such as Vitual Cord Protocol (VCP) [21], [22], which is a virtual location based protocol. The main specificity of VCP is that virtual and physical neighbors in VCP are the same. The construction algorithm is as follows:

1) A global ID is assigned to each node, constructed as the node's IP address[2]. Let's call it *GlobalLocalityID*.
2) In each building/house, VCP is used to assign a local, relative ID to each local IoTD. Let's call it *LocalLocalityID*.
3) In each building/house, one IoTD is chosen as a BC connection point (e.g. level-1 CH) for the IoTDs located in the same cluster. Flagged as "CH", each CH is labeled by a one-digit CHAC indicating the CH's logical position in the forest. The CH's CHAC value is communicated to its managed IoTDs, which then set their CHAC values to match that of the CH. Using a locality table, the CH maps an IoTD's *GlobalLocalityID* to its corresponding *LocalLocalityID*. Using a VCP table, the CH routes traffic towards its managed IoTDs.
4) Considering each couple of adjacent buildings/houses (clusters), one IoTD is chosen from any of those clusters s.t. the chosen IoTD has a *GlobalLocalityID* closer (having the same L-most significant bits, where L is determined based on each application's needs) to the managing level-1 CH *GlobalLocalityID* of any of the two adjacent clusters. The chosen IoTD is flagged as "level-2 CH". All level-2 CHs are labeled with a one-digit CHAC and this digit is appended to the CHAC of the managed level-1 CHs and level-0 IoTDs.
5) Additional IoTDs are selected as CHs to construct subsequent levels. Each such IoTD is chosen from a set of adjacent clusters based on how close their *GlobalLocalityID* is to their managing CH's *GlobalLocalityID* (having the same L-most significant bits). A chosen CH is assigned a one-digit CHAC that is also appended to all its managed CHs and IoTDs. The process is repeated for every forest level added and we end up with a partitioned ring of CHs, some having local IoTD clusters, as shown in figure 4.

At the end of the execution, we obtain a ring with CHs having various CHAC lengths to indicate their level within the structure. Each level-1 CH has a locality table, a VCP table and a set of managed IoTDs. A CH is identified by a *GlobalLocalityID* and a CHAC, while a level-1 CH is additionally identified by a *LocalLocalityID*. Each IoTD has a VCP table and is identified by a *GlobalLocalityID*, a *LocalLocalityID*, as well as the same CHAC as that of its CH.

[2]IPv6 address is recommended for this algorithm.

It's worth mentioning that, a CH role changes on a timeout round-robin basis to allow a fair chance for all the local IoTDs to act as CHs. If a CH becomes inactive, goes offline, proves incapable of managing its assigned IoTD cluster, or times out as a CH, it can undergo one of two processes: demotion as an IoTD node and replacement by another CH from the IoTD nodes, or reassignment with a lower management load. The latter process includes splitting the CH's managed cluster by reapplying VCP. The CH still manages a sub-cluster while the other sub-cluster is assigned to another CH (a promoted IoTD node from that sub-cluster), thereby sharing the load.

The cost of building the architecture includes the cost for assigning the *GlobalLocalityID* to the nodes and the per cluster construction of VCP and the locality table, as well as the cost of promoting CHs to build the levels. The cost of maintaining the structure (to cover future splits as the system grows) includes the cost of rebuilding VCP as well as the associated CH promotion. Thresholds may be assigned to cluster sizes, akin to the SSD's MUT, s.t. all level-0 clusters may split once a predefined number of clusters exceeds the threshold (in order to keep all cluster sizes comparable), resulting in adding more levels to the hierarchy.

To achieve consensus, a Tx needs to propagate from an IoTD Tx party towards the other party in order to obtain the latter's public key. Assuming both Tx parties can exchange their IP addresses in advance, a Tx generated by an IoTD party propagates towards its topmost parent CH in $O(1)$ using CHAC. Then a Tx propagates towards the topmost CH parent of the other Tx party, whose *GlobalLocalityID* closest to the other Tx party's *GlobalLocalityID*, in $O(maxprop)$. Then the Tx propagates in $O(structure's\,levels\,times\,branching\,factor)$ to compare the other Tx party's *GlobalLocalityID* to the *GlobalLocalityID* of each CH located in the subtree leading to the other Tx party. Then the Tx propagates in $O(\sqrt{(local\,IoTDs)}-1)$ using VCP to reach the other Tx party (within its cluster). Then the Tx propagates in $O(1)$ (using CHAC) to reach the topmost level CHs which verify the Tx in $O(maxprop)$. Also, a block is verified by the topmost CHs in $O(maxprop)$. Thus consensus takes $O(2)$ plus $O(structure's\,levels\,times\,branching\,factor)$ plus $O(\sqrt{(local\,IoTDs)}-1)$ plus three times $O(maxprop)$, worst case.

Assuming both Tx parties are located within the same cluster, consensus takes $O(\sqrt{(local\,IoTDs)}-1)$ to reach the other Tx party, then $O(1)$ to reach the topmost level CHs to verify the Tx in $O(maxprop)$. Also a block is verified by the topmost CHs in $O(maxprop)$. Thus consensus takes $O(\sqrt{(local\,IoTDs)}-1)$ plus $O(1)$ plus two times $O(maxprop)$, best case.

D. Option 4 - Improved Ring of Rings

Design option 4 is another variation of the previous architecture where CHs' CHAC is communicated to the IoTD nodes.

Assuming a level-1 CH manages a table that maps CHAC to *GlobalLocalityID* for all level-1 CHs and assuming both Tx parties can exchange their IP addresses as well as their CHAC in advance, identifying the other Tx party's level-1 CH is by comparing the other Tx party's *GlobalLocalityID* to the closest *GlobalLocalityID* that belongs to a level-1 CH, in $O(Number\,of\,level-1\,CHs)$. Then a Tx generated by an IoTD party propagates towards the other Tx party's level-1 CH in $O(1)$ using CHAC. Then the Tx propagates in $O(\sqrt{(local\,IoTDs)}-1)$ to reach the other Tx party within it cluster. Then the Tx propagates in $O(1)$ (using CHAC) to reach the topmost level CHs, which verify the Tx in $O(maxprop)$. Also a block is verified by the topmost CHs in $O(maxprop)$. Thus consensus takes $O(2)$ plus $O(Number\,of\,level-1\,CHs))$ plus $O(\sqrt{(local\,IoTDs)}-1)$ plus two times $O(maxprop)$, worst case.

Best case is as in design option 3: when both Tx parties are in the same cluster, consensus is in $O(\sqrt{(local\,IoTDs)}-1)$ plus $O(1)$ plus two times $O(maxprop)$.

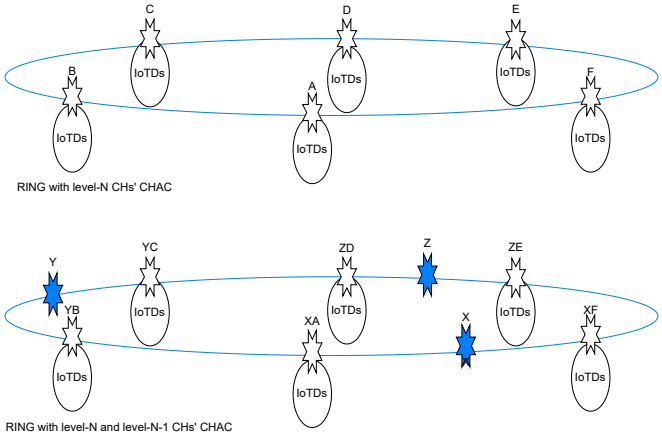

Fig. 4. Ring of Rings

IV. Comparison of Proposed Designs

The proposed designs improve consensus time when compared to flat and two-tiered architectures. This improvement is attributable to the multi-tiered design, which increases scalability and reduces the number of topmost CHs involved in the consensus process to a logarithmically bounded reduction of the CHs directly managing IoTDs. Further improvement is expected with the addition of more levels.

Based on the asymptotic analysis, SSD's consensus efficiency outperforms that of the other proposed designs. Similarly, Improved Ring of Rings' consensus efficiency outperforms that of Ring of Rings. While efficiency criteria are crucial, the decision to adopt one architecture over another depends on whether the application requires a given level of flexibility. In cases where the architecture must continuously adapt to IoTDs joining and leaving, options such as SSD, Ring of Rings and Improved Ring of Rings may be suitable. Conversely, if the construction phase can occur before conducting BC operations, the Partitioned architecture might be the appropriate choice. However, flexibility comes with

associated maintenance costs that should also be taken into account.

Based on the specific needs of an application, requirements might include conditions related to node proximity. In such cases, choosing the Ring of Rings or Improved Ring of Rings architectures, could be appropriate. However, if maintaining comprehensive CHAC information for all IoTDs proves challenging or excessively costly, opting for the Ring of Rings architecture might be preferable over the Improved Ring of Rings, despite the latter having a more efficient consensus.

Table I summarizes the respective advantages of each proposed architecture design concerning the earlier mentioned criteria.

It is worth mentioning that, in all proposed designs, while Tx and block verifications occur at level N (the topmost level), Tx verifications can also be configured to take place at a level closer to IoTDs if a shared CH parent for both Tx parties is located at that level and if Tx tampering is a concern. Configured in this manner, the multi-tiered designs improve performance by decoupling Tx verifications from block verifications, thereby distributing block processing and packet traffic overhead across all levels of the architecture, as shown in Fig. 5.

	Consensus Effectiency	Structure Flexibility	Maintenance Efficiency	Node Proximity
SSD	x	x		
Partitioned			x	
Ring of Rings		x		x
Improved Ring of Rings		x		x

TABLE I
PROPOSED DESIGNS COMPARISON TABLE

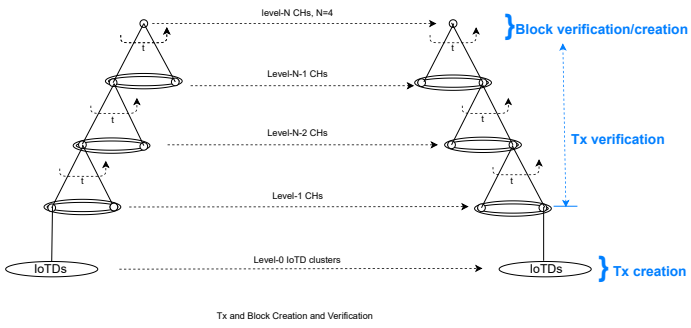

Fig. 5. Proposed Designs with N=4.

V. PERFORMANCE AND SIMULATION RESULTS

In order to showcase the potential of the multi-tiered approaches and assess the efficiency of the consensus mechanism applied, we conducted simulations specifically for the SSD architecture and algorithm (option 1). We implemented two sets of simulations:

1) The first set of simulations is to compare the efficiency of flat, two-tiered and multi-tiered models: for 10 runs, processing 100 transactions with 260 IoTDs (1.5% fluctuations).
2) The second set of simulations is to evaluate colocality. Thus we simulated 2 CHs, 4 CHs and 8 CHs multi-tiered builds for 10 runs, processing 500 transactions with 260 IoTDs (1.5% fluctuations).

All simulations are programmed in Python3 where PoW, Tx creation, propagation delays and block and Tx verifications are modeled. Our multi-tiered model outperforms flat model (with only 6% of nodes as block-generating nodes) and two-tiered model in average Tx latency and throughput as shown in Fig. 6. Increasing the percentage of block-generating nodes in flat model impacts the model performance exponentially as shown in Fig. 7.

Increasing colocality improved the multi-tiered model performance and throughput as depicted in Fig. 8. Leveraging colocality, for 90% of the runs, consensus time is reduced by almost 17% for colocal Tx.

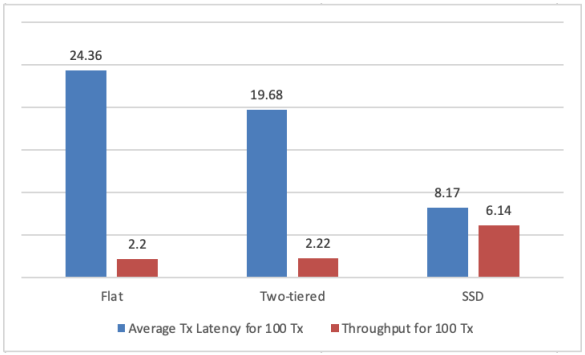

Fig. 6. Average Tx Latency and Throughput for 100 Tx in BC Models.

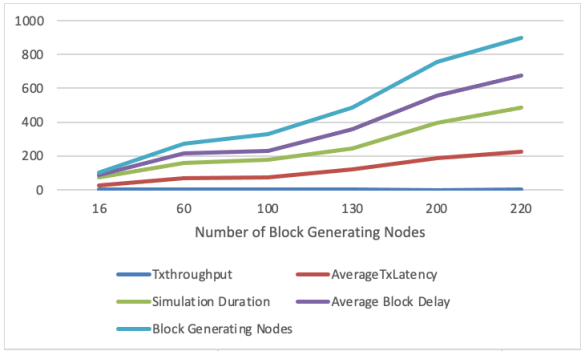

Fig. 7. Flat Model 100 Tx - Average Delays And Block Generating Nodes.

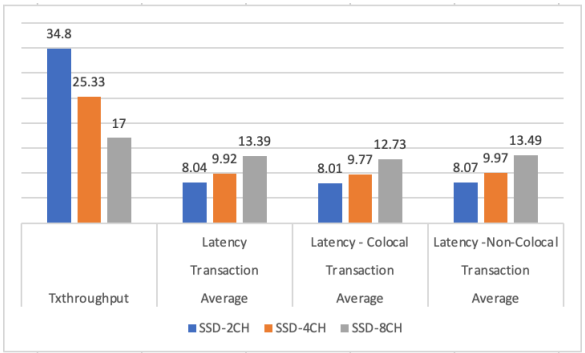

Fig. 8. Comparison of the SSD BC builds for 500 Tx.

CONCLUSION

In this paper, we present four multi-tiered BC architecture designs aiming at reducing consensus time and increasing scalability when applied for IoT context: i) Self-Scalable Dynamic (SSD) architecture, ii) Partitioned architecture, iii) Ring of Rings architecture and iv) Improved Ring of Rings architecture. Central to these architectures are mechanisms for node lookup and message routing to easily locate nodes and route messages within each design. Additionally, a proposed consensus mechanism is incorporated, adaptable to each proposed design.

We conduct a comparative analysis of the proposed designs and provide simulations for one of the proposed architectures, specifically the SSD Architecture and Algorithm (design option 1). The simulations serve as a proof of concept for the multi-tiered approaches when integrated with the proposed consensus mechanism.

Based on the simulations, the multi-tiered design concept has demonstrated its viability as a solution for implementing BC into the IoT domain. The performance trends indicate that our proposed multi-tiered models not only outperform flat and two-tiered models in terms of efficiency and scalability but are also expected to further improve with an increasing number of implemented IoTDs.

The results obtained from the simulations of design option 1 represent just one potential outcome within the extensive scope of results achievable through multi-tiered architecture designs. The other proposed multi-tiered designs may yield different outcomes, influenced by the performance of the consensus mechanism and its adaptability to each specific design.

Nonetheless, while performance remains a crucial consideration, it should not overshadow the broader scope of factors influencing the choice of a design option for a given application. Beyond performance metrics, factors such as construction flexibility, maintenance costs, and node proximity play pivotal roles in making a well-informed decision.

In essence, the comprehensive spectrum of scalable, application and algorithm-agnostic multi-tiered design alternatives unveiled in this study not only demonstrates their superiority but beckons as a compelling solution to significantly boost the implementation of efficient BC within the dynamic framework of the IoT ecosystem.

REFERENCES

[1] M. Al-Shareeda, M. Ali and S. Manickam, "The blockchain internet of things: review, opportunities, challenges, and recommendations," Indonesian Journal of Electrical Engineering and Computer Science, vol. 31, pp. 1673-1683, doi 10.11591/ijeecs.v31.i3.pp1673-1683, September, 2023.

[2] S. Singh and S. Duggal, "Challenges of integration of blockchain into internet of things (IoT): A survey," AIP Conference Proceedings, doi 10.1063/5.0118864, December, 2022.

[3] A. Dorri, S. Kanhere, R. Jurdak and P. Gauravaram, "Lsb: A lightweight scalable blockchain for iot security and privacy," Journal of Parallel and Distributed Computing, 134, http://dx.doi.org/10.1016/j.jpdc.2019.08.005, 2017.

[4] Y. Yu, "Analysis of POW in Bitcoin and POS in Peercoin," Highlights in Science, Engineering and Technology, vol. 39, p. 784–788, doi 10.54097/hset.v39i.6645, April, 2023.

[5] M. Alharby and A. Van Moorsel, "BlockSim: An Extensible Simulation Tool for Blockchain Systems," Front. Blockchain, Sec. Financial Blockchain, vol. 3, https://doi.org/10.3389/fbloc.2020.00028, June 2020.

[6] A. Albulayhi and I. Alsukayti, "A Blockchain-Centric IoT Architecture for Effective Smart Contract-Based Management of IoT Data Communications," Electronics, vol. 12, pp. 2564, June, 2023.

[7] V. Buterin, "A Next-Generation Smart Contract and Decentralized Application Platform," Ethereum_Whitepaper, 2021.

[8] L. Zhang, J. Ron, B. Baudry and M. Monperrus, " Chaos Engineering of Ethereum Blockchain Clients," Distributed Ledger Technologies: Research and Practice, doi 110.1145/3611649, August, 2023.

[9] S. Tanwar, N. Gupta, J. Tomar, K. Kumar, S. Treľová and I. Ivanochko, "A Review on Blockchain Smart Contract Applications," doi 10.1007/978-3-031-37164-6_13, isbn 978-3-031-37163-9, pp. 175-187, September 2023.

[10] S. Badruddoja, R. Dantu, Y. He, M. Thompson, A. Salau and K. Upadhyay, "Making Smart Contracts Predict and Scale," doi 10.1109/BCCA55292.2022.9922480, pp. 127-134, September 2022.

[11] T. H. Cormen, C. E. Leiserson and R. L. Rivest and C. Stein, "INTRODUCTION TO ALGORITHMS," MIT Press, McGraw-Hill, pp. 434–454,1990.

[12] S. Yildirim, "Top Machine Learning Algorithms for Clustering," https://towardsdatascience.com/top-machine-learning-algorithms-for-clustering-a09c6771805, April 2020.

[13] Z. Huang, "Extensions to the k-Means Algorithm for Clustering Large Data Sets with Categorical Values," Data Min. Knowl. Discov., Volume 2, doi 10.1023/A:1009769707641, pp. 283–304, September 1998.

[14] H. Karimi and O. Medhati and H. Zabolzadeh and A. Eftekhari and F. Rezaei and S. Dehno and A. jamalpoor, "Implementing a Reliable, Fault Tolerance and Secure Framework in the Wireless Sensor-actuator Networks for Events Reporting," Procedia Computer Science, Volume 73, doi 10.1016/j.procs.2015.12.007, December 2015.

[15] I. Eyal and I. Keidar and R. Rom, "Distributed data clustering in sensor networks," Distributed Computing, Volume 24, doi 10.1007/s00446-011-0143-7, pp. 207–222, December 2011.

[16] S. Demian Lerner, "Uncle Mining, an Ethereum Consensus Protocol Flaw," https://bitslog.wordpress.com/2016/04/28/uncle-mining-an-ethereum-consensus-protocol-flaw/, 2023.

[17] U. Brandes, D. Delling, M. Gaertler, R. Gorke, M. Hoefer, Z. Nikoloski and D. Wagner, "On Modularity Clustering," https://algo.cs.uni-frankfurt.de/ mhoefer/05-07/Brandes07ModularityJ.pdf, 2007.

[18] R. Elsaadany and G. Bégin, "A Blockchain Node Organizing and Auto-Scaling Architecture for the Internet of Things," 2023 Fifth International Conference on Blockchain Computing and Applications (BCCA), Kuwait, Kuwait, 2023, pp. 36-43, doi: 10.1109/BCCA58897.2023.10338882.

[19] Galuba, W., Girdzijauskas, S. (2009). Distributed Hash Table. In: LIU, L., ÖZSU, M.T. (eds) Encyclopedia of Database Systems. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-39940-9_1232

[20] I. Stoica and R. Morris and D. Karger and M. Kaashoek and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," ACM SIGCOMM Computer Communication Review, vol. 31. 31. 10.1145/964723.383071, 2001.

[21] F. Ghofrane, "A distributed and flexible architecture for Internet of Things," Elsevier B.V, Procedia Computer Science, vol. 73, pp. 130–137, 2015.

[22] A. Awad, R. German and F. Dressler, "Exploiting Virtual Coordinates for Improved Routing Performance in Sensor Networks," doi 10.1109/TMC.2010.218, Mobile Computing, IEEE Transactions on, vol. 10, pp. 1214–1226, 2011.

[23] J. Krajewski, J. Lozano, J. Driver, E. Escandon, S. Kumar, S. Malatini and J. Lozano Hinojosa, "PASTRY: the third generation of peer -to -peer networks," January 2006.

[24] A. Rowstron, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," doi 10.1007/3-540-45518-3_18, Lecture Notes in Computer Science, isbn 978-3-540-42800-8, vol. 2218, October 2002.