Bringing secure Bitcoin transactions to your smartphone

Davide Frey
Inria Rennes
France
davide.frey@inria.fr

Marc X. Makkes
Vrije Universiteit Amsterdam
The Netherlands
m.x.makkes@vu.nl

Pierre-Louis Roman
University of Rennes 1 – IRISA – ESIR, France
pierre-louis.roman@irisa.fr

François Taïani
University of Rennes 1 – IRISA – ESIR, France
francois.taiani@irisa.fr

Spyros Voulgaris
Vrije Universiteit Amsterdam
The Netherlands
spyros@cs.vu.nl

ABSTRACT

To preserve the Bitcoin ledger's integrity, a node that joins the system must download a full copy of the entire Bitcoin blockchain if it wants to verify newly created blocks.

At the time of writing, the blockchain weights 79 GiB and takes hours of processing on high-end machines. Owners of low-resource devices (known as *thin nodes*), such as smartphones, avoid that cost by either opting for minimum verification or by depending on full nodes, which weakens their security model.

In this work, we propose to harden the security model of thin nodes by enabling them to verify blocks in an adaptive manner, with regards to the level of targeted confidence, with low storage requirements and a short bootstrap time. Our approach exploits *sharding* within a *distributed hash table* (DHT) to distribute the storage load, and a few additional hashes to prevent attacks on this new system.

Categories and Subject Descriptors

D.4.7 [**Organization and Design**]: Distributed systems; E.1 [**Data structures**]: Distributed data structures

Keywords

Blockchain, Bitcoin, Sharding, Distributed hash table

1. BACKGROUND

Introduced in 2008, Bitcoin [11] has emerged as the first widely-deployed decentralized global cryptocurrency. It has triggered the creation of numerous similar projects, such as Ethereum [16], or Namecoin [5]. The current market capitalization of Bitcoin surpasses 9 billion USD[1] and a significant fraction of it is used in the overall 200,000 daily transactions. Bitcoin is fully decentralized and requires no central bank or authority. It relies solely on the distributed maintenance of a data structure called *blockchain*.

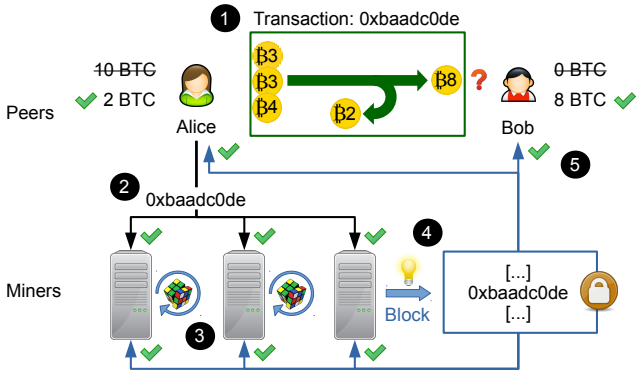

Figure 1: A new transaction (1), is advertised to miners (2), who verify its integrity and insert it into a block by solving a cryto-puzzle (3). This block (4) (which forms a chain with the previous blocks, not shown) is then broadcast to the rest of the network (5) which in turn verifies its integrity to publicly acknowledge the transaction between Bob and Alice.

The blockchain is the core data structure introduced by Bitcoin to prevent the acceptance of fraudulent transactions. It essentially constitutes a distributed ledger which stores an ever growing list of transactions in a secure manner. All transactions stored in the blockchain are considered immutable and valid.

Because of its immutability, a blockchain can be abstracted as a transactional system that enables a consensus to form within its participants. This consensus holds unique probabilistic properties and can thus be leveraged as a basic building block for adaptive middlewares that offer both deterministic and probabilistic consensus.

As its name suggests, a blockchain is a sequence of *blocks*. Each block contains a number of *transactions*, as well as a cryptographic hash of the previous block, which links the two blocks and effectively form a chain. Each transaction, in turn, contains a set of *inputs*, i.e., the coins the payer spends (for instance three coins of 3, 3, and 4 BTC belonging to Alice in Figure 1, Step **1**), and a set of *outputs*, i.e., the coins delivered to the payee(s) (8 BTC delivered to Bob, and 2 BTC returned as change to Alice). Finally, each *coin*

[1]https://bitinfocharts.com, 2016-09-01

ARM 2016, December 12-16, 2016, Trento, Italy

DOI: http://dx.doi.org/10.1145/3008167.3008170

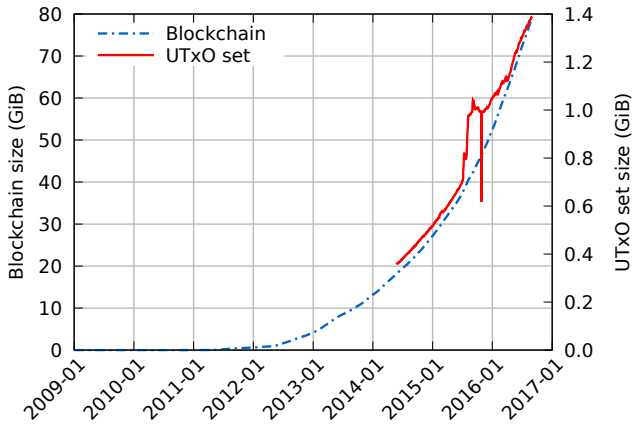

Figure 2: Both the blockchain and the UTxO set have almost quadrupled in size in the past two years.

represents an arbitrary amount of money, measured at a granularity of 10^{-8} BTC (an amount known as *1 satoshi*). A coin is generated through a transaction, is immutable for its entire lifespan, and can only be spent once.

To *mine* (i.e., create) a block containing several transactions (Step **2**), a node has to solve a cryto-puzzle called *Proof-of-Work* (PoW) (Step **3**). Nodes who actively try to solve this puzzle are called *miners*. The difficulty of solving the PoW is automatically adjusted every two weeks, so that a block is generated on average every 10 minutes. The newly mined block (Step **4**) is then propagated to the rest of the network (Step **5**), who can check its validity. A consensus rule is instated to resolve forks and discard blocks that are not part of the chain with the most work. This rule ensures that miners converge in practice to a unique valid view of the blockchain that prevents history rewrites and double spending of coins. The difficulty to create a block, combined with this consensus rule, provides immutable valid transactions.

Before a miner inserts a transaction into a block (Step **2**), it checks the transaction's validity. Once a transaction has been added to the blockchain, all its inputs are considered spent, and only its outputs are spendable. A transaction is filed as valid if all its inputs exist as outputs of past transactions in the blockchain and have not yet been spent. To speed up this verification process, nodes keep track of unspent coins (and hence spendable) in a local data structure known as the set of *Unspent Transaction Output* (UTxO).

In addition to miners, two more types of nodes are involved in the Bitcoin ecosystem: *full nodes* and *legacy thin nodes*[2]. Both types of nodes implement *Bitcoin wallets*, storing a user's cryptographic credentials and owned coins, compiling transactions when the user makes a payment, and checking the validity of transactions when the user receives a payment. The sole difference between full and legacy thin nodes is that full nodes pay the full price for being able to *locally* verify transactions' validity, while legacy thin nodes outsource that verification to the former, effectively trading their level of trust for lower resource expenditure.

[2]What is usually called a thin node in the current Bitcoin system, or specifically a *Simple Payment Verification* (SPV) client, is denoted legacy thin nodes in this paper to have a clear distinction with the secure thin nodes we propose.

The price full nodes have to pay in terms of resources is indeed high. A full node joining the network has to download the entire blockchain (427 K blocks, 79 GiB)[3] and parse it sequentially to build the UTxO set (41.4 M coins, 1.39 GiB)[4] from scratch. This simple process consumes excessive bandwidth, computational resources and time, while it is not scalable with respect to storage requirements. Figure 2 illustrates the rapid growth of the sizes of both the blockchain and the UTxO set between January 2009 and August 2016, and demonstrates the need to lower the bandwidth and storage requirements for resource-constrained devices.

Legacy thin nodes, on the other hand, avoid these resource costs entirely by skipping the task of storing the blockchain and the UTxO set. As a consequence, however, they are incapable of validating their transactions *locally*, having to rely on full nodes for outsourced validation. Clearly, this diminishes the level of security experienced by legacy thin nodes. In perspective, this constitutes a severe shortcoming of the current Bitcoin design, as the massive spread and enormous convenience of smartphones suggest that thin nodes will outnumber their full node counterparts by far, once the wider population embraces digital currency. Even worse, in the longer term it will severely undermine the overall public acceptance of the Bitcoin system.

Our work focuses precisely on the aforementioned gap in Bitcoin's design. More specifically, we propose an architecture that targets legacy thin nodes, and enables them to validate their transactions *locally*, without having to store the blockchain or the UTxO set. This essentially raises legacy thin nodes' trust level close to that of full nodes, without requiring prohibitive amounts of resources.

We present a coarse-grained view of our envisioned solution in Section 2 and its details in Section 3. We further elaborate on the benefits our solution has to offer in Section 4, while we present related work in Section 5. Finally, we conclude this paper in Section 6.

2. ARCHITECTURE OVERVIEW

Our ultimate goal is to raise legacy thin nodes' trust level close to that of full nodes, by letting them verify their transactions *locally*, without the need to arbitrarily trust other nodes. As thin nodes are not expected to keep track of the UTxO set, our solution focuses on providing them with the alternative option of acquiring the UTxO set when needed.

We go in fact one step further by allowing thin nodes to only acquire the specific *part(s)* of the UTxO set that is required to validate one or more input coins. Thin nodes that can validate input coins are called *secure thin nodes*. In line with our goal of making Bitcoin practically accessible to smartphones and resource-constrained devices, it would be unrealistic to expect such devices to download a UTxO set as large as 1.39 GiB in compressed form. Especially since the UTxO set will likely continue its rapid growth.

Our intuition is as follows: we split the UTxO set into *shards* that we store within a *distributed hash table* (DHT) that secure thin nodes can query whenever they need to validate the inputs of a transaction. To realize this intuition in practice, two problems need to be addressed: first, the

[3]Blockchain size dataset as of 2016-09-01: https://blockchain.info/charts/blocks-size.

[4]UTxO size dataset as of 2016-09-01: https://statoshi.info/dashboard/db/unspent-transaction-output-set.

shards should ideally be of uniform size to ensure the DHT is well-balanced; and second, appropriate care should be taken to guarantee the integrity of the shards against malicious nodes. To address the first problem, we partition the UTxO set into 2^k shards, S_0, S_1, ..., S_{k-1} by using the first k bits of the coin's hash to determine which shard the coin should be allocated to. To address the second problem, we then compute the SHA-256^2 hashes of the shards, H_0, H_1, ..., H_{k-1}, which we compute after having sorted the coins within each shard. Finally, we produce a SHA-256^2 hash of all shard hashes, which we call the *UTxO hash*, H_{UTxO}. We then include the 32-byte long UTxO hash in the block of the official Bitcoin blockchain.

When a secure thin node wants to validate an input of an incoming transaction, it hashes it, and fetches from the DHT the UTxO shard indexed by the first k bits of the hash. It then simply checks that this input is contained in this shard.

The crucial point is the way a secure thin node validates the legitimacy of a shard it fetches. To do so, it also fetches the entire list of shard hashes, H_0, H_1, ..., H_{k-1}, and verifies the hash of that list against the H_{UTxO} value stored in the blockchain. Once it has validated the list of hashes, it can use the hashes in this list to verify the integrity of the shard(s) it queries.

The last missing piece of the puzzle is the way secure thin nodes get hold of the H_{UTxO} from the blockchain. Nodes download at some point the current block from a "trusted" source, and thereafter keep receiving subsequently generated blocks. This way, their confidence in the received H_{UTxO} is the same as their confidence in the blockchain itself.

Starting to follow the blockchain at some arbitrary starting block rather than at the "genesis block" makes a secure thin node's trust model weaker than that of a full node. However, if a newly joined secure thin node waits for a number of blocks to be generated, checks that they are all valid, that they adhere to the proof-or-work difficulty, and that they are correctly timestamped, it has an extremely strong indication that it is indeed following the legitimate blockchain, as it would have been practically impossible for a set of malicious nodes to generate several blocks of the right difficulty. For added confidence, a newly joined node may also make a trivial transaction (e.g., transfer 1 satoshi to itself), wait to see it in a block, and wait for a number of blocks thereafter before it trusts the blockchain. This will rule out precomputed blocks in a replay attack.

Finally, UTxO shards have to be stored somewhere so that they are readily available to secure thin nodes that need them. For that, we propose the use of a DHT which can be executed on any nodes, although we expect that full nodes and secure thin nodes will typically be part of it. Full nodes will further keep a copy of the entire set of shards, while each secure thin node will keep a small subset.

3. PROPOSED SYSTEM

Our proposed design involves five distinct roles, namely miners, full nodes, secure thin nodes, legacy thin nodes, and DHT nodes. Some of these roles can be executed by the same physical nodes, for instance miners, full nodes, or secure thin nodes would typically also be part of the DHT.

Note that our proposed design is completely transparent from the point of view of full nodes and legacy thin nodes, which will continue to function precisely the way they do.

In this section, we present the data structures needed to obtain secure thin nodes and the precise operation of our design, split by roles.

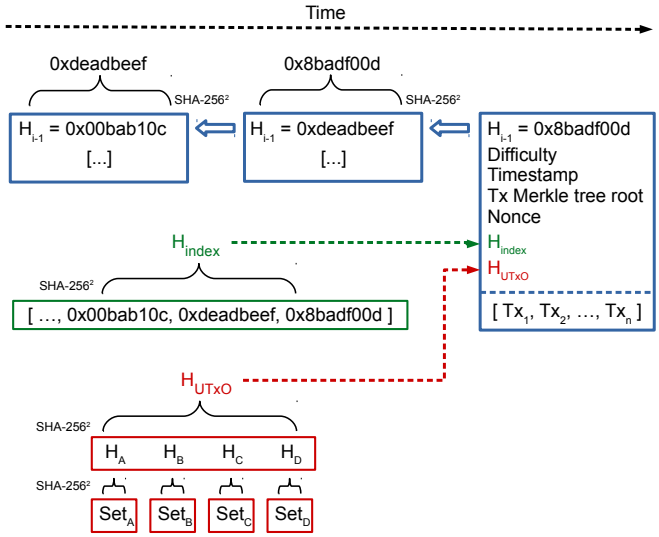

Figure 3: The block index hash and the UTxO hash are added to the block header.

3.1 Data structures

We introduce two data structures to guarantee the integrity of our design against malicious nodes. The first concerns the integrity of UTxO shards, while the second guarantees the integrity of blocks distributed through the DHT.

As explained in Section 2, we partition the UTxO set into 2^k shards, where each coin's placement is determined by the first k bits of the coin's hash. To guarantee the shards' integrity, we define a two-layer set of hashes, as highlighted in red in Figure 3. The lower level of hashes consists of one hash per shard, $H_i = HASH(S_i)$, $\forall i \in \mathbb{Z}_{[0,k-1]}$. The upper level consists of just one hash of all H_i, that is, $H_{\text{UTxO}} = HASH(\cup_{i=0}^{k-1} H_i)$. The hashing function $HASH$ is the one used in Bitcoin, i.e., $HASH(x) = \text{SHA-256}(\text{SHA-256}(x))$, noted SHA-256^2 throughout this document.

The second structure we use concerns the integrity of blocks of the blockchain. To freely access any section of the ledger, a secure thin node should be able to retrieve arbitrary blocks from the DHT. A block is indexed in the DHT by its own hash value. Therefore, we maintain an array of all blocks' hashes, which we call the *index*. This array is hashed each time a new block is created to obtain a hash of the index, H_{index}, highlighted in green in Figure 3.

Both 32-byte hashes, H_{UTxO} and H_{index}, are placed in block headers. The blockchain acts as an alteration-proof communication channel, ensuring in effect the broadcast of the correct values of H_{UTxO} and H_{index}, every time a new block is created.

3.2 Miners

Miners constitute the core of the Bitcoin network. They try to solve the proof-of-work computational puzzle in order to create new blocks, and profit from the monetary compensation associated with it. More specifically, they receive from wallet nodes (i.e., full or legacy/secure thin nodes) announcements about new transactions that should be in-

cluded in future blocks. Each miner picks an arbitrary set of pending transactions, typically the maximum number that can fit in one block with a preference for transactions offering a higher mining fee, and generates the SHA-256^2 hash of the block for billions of arbitrary 32-byte values in the *nonce* field. A miner stops generating hashes once one of them happens to be below a predetermined value, known as the mining difficulty level. At this point, a new block has been created, and the miner that found it announces it to the entire set of miners. Each miner receiving a newly created block validates the block's integrity. That is, it makes sure that (i) the block's SHA-256^2 is indeed lower than the difficulty threshold, (ii) all transactions in it are legitimate (i.e., if all input coins are in the UTxO set and are signed by their claimed owners), and that (iii) the miner that created it has allocated the correct amount to his own wallet (i.e., the number of bitcoins offered per created block, plus the fees of all included transactions). If the block is valid and extends the blockchain, the miner starts working on creating the next block to be linked to the newly received block.

In the context of our proposed design, miners should also include two 32-byte fields in each block header: the H_{UTxO} and H_{index} hashes defined in Section 3.1.

Note that H_{index} cannot include the hash of the current block, as this would imply a circular dependency between the two. Therefore, H_{index} contains the hash of the index including all blocks up to and including the previous one.

Similarly, H_{UTxO} refers to the UTxO set of the previous block. That is, the UTxO set right before any transactions of the current block have been applied. The reason for this has to do with performance. If the current block's transactions had to be included in the H_{UTxO}, as miners keep receiving new transactions while trying to mine the next block, they would have to internally keep track of updates to the UTxO set, its shards, its shards' hashes, and the hashes' aggregate hash each time a new transaction is received. This would add an unnecessary burden to the mining process. Instead, including the previous block's H_{UTxO}, which is a fixed hash computed once, has negligible computational implications for the mining process. Additionally, it also benefits nodes when they receive new blocks. They can instantly verify the H_{UTxO} present in the block, without having to take into account the transactions included in the block.

Finally, in addition to disseminating a block that has been newly created, miners should now also upload the block, and the respective UTxO shards to the DHT.

3.3 Secure thin nodes

A user willing to verify an incoming payment may now use a secure thin node in her smartphone, instead of the less secured SPV client. Upgrading to a secure thin node enables a user to verify the validity of transactions contained in a block by querying the UTxO set built by other nodes and stored in the DHT. More details on the security aspect of our proposal on secure thin nodes can be found in Section 4.1.

When a secure thin node joins the system, it bootstraps by downloading from another node: (i) the last few blocks (e.g., 6), (ii) the list of UTxO shard hashes, and (iii) the list of block hashes.

The last few blocks are used to resolve blockchain forks as well as to extract H_{UTxO} and H_{index}. These hashes are used to verify the integrity of the data retrieved from the DHT.

The list of UTxO shard hashes is used to securely verify transactions contained in newly mined blocks. Once retrieved, the list is hashed and compared with the H_{UTxO} present in the latest received block. To verify a transaction, a secure thin node identifies its inputs, finds which UTxO shard each input should belong to, and queries the DHT for the shards it needs. It then ensures the validity of each shard by comparing its hash against the one it downloaded before. If an input is indeed in the shard, it is valid.

The list of block hashes is used to query the DHT for blocks for historical purposes. Full nodes verify the whole blockchain as part of their bootstrap, which enables them to know for sure which blocks are part of the blockchain. Secure thin nodes avoid this long process, and therefore need another way to determine which blocks compose the blockchain. In our system, we make the list of block hashes downloadable and verifiable. Once received, this list is hashed and compared against the H_{index} hash contained in the latest block. A correct match indicates that this list has been agreed upon by a consensus of miners and is thus deemed valid by a secure thin node.

Finally, when a secure thin node receives a new block, it can verify its headers as well as the transactions whose inputs belong to the UTxO shards it possesses. Missing shards can be queried to verify other transaction inputs. Additionally, secure thin nodes can locally update their UTxO shards by applying to them each relevant transaction inputs and outputs from the newly received block.

3.4 DHT nodes

To handle the need for secure thin nodes to retrieve arbitrary shards, we propose to federate the storage space of a large number of nodes by using a DHT to collectively store and provide shards. Note that, in principle, DHT nodes can be Bitcoin agnostic; even nodes without any Bitcoin-related role can be part of the DHT.

As secure thin nodes may need to access past blocks' shards, the DHT should maintain a shard history of at least the h most recent blocks. In order to index all these shards, DHT indexes should be a combination of the relevant block's hash, and the shard index $(0 \dots k-1)$ within that block's UTxO set. Such an index can be provided by $HASH(H_{\text{block}} \cup i)$, where H_{block} is the relevant block's hash and i is the specific shard's index.

In addition to shards, the DHT should also help in retrieving arbitrary blocks of the blockchain, which will otherwise become a bottleneck as the blockchain keeps growing. There is no need for a separate DHT. Instead, blocks can be stored in the same DHT, indexed by their block hash.

Clearly, our DHT key space is $[0, 2^{256}-1]$, accepting SHA-256 hashes as keys. The selection of the specific DHT, however, is not relevant for the correct operation of our system, as the $\langle \text{key} \rightarrow \text{value} \rangle$ mapping is a standard functionality for all DHTs. The only differentiating points are the ability of DHT nodes to arbitrarily pick their contributed storage, and the ability of the DHT to cater for a sufficient level of replication, but these are implementation design decisions that are beyond the scope of this paper.

4. PRELIMINARY ANALYSIS

Our secure thin nodes offer a trade-off between existing full nodes and Simplified Payment Verification (SPV) clients (legacy thin nodes), both in terms of security model and in terms of storage and bandwidth consumption.

4.1 Security model of secure thin nodes

Secure thin nodes offer a security model that approaches that of full nodes while being significantly stronger than that of legacy SPV clients.

Full nodes verify every operation. When a full node bootstraps, it downloads and verifies the entire chain, while building its own UTxO set. This enables full nodes to operate without having to trust any other node. An SPV client, on the other hand, must connect to a full node, which will forward only the blocks that contain the transactions that directly concern the client. This limited information makes it impossible for SPV clients to detect fraudulent blocks and transactions and forces them to rely entirely on their associated full node. A malicious full node could easily trick SPV clients simply by omitting to forward blocks, or by forging blocks with fraudulent transactions.

Secure thin nodes strike a balance between these two extremes by verifying only a portion of the blockchain instead of all of it. Secure thin nodes start the verification at some arbitrary block, and they continue verifying blocks thereafter. After having verified a number of blocks generated with the correct difficulty and correct timestamp , they can be practically certain that they are following the legitimate chain, as it would have been computationally close to impossible for a set of malicious nodes to produce blocks of such difficulty. Thus, they get hold of the H_{UTxO}, which enables them to verify the specific shards they need, to validate transactions when receiving payments. This grants them a trust level close to that of full nodes.

However, secure thin nodes generally validate the hashes of new blocks, and only a subset of the included transactions (those that are directly relevant to them). Thus, in a well orchestrated attack a secure thin node T could be presented with a fake block that contains a *valid* transaction paying money to T, but that deliberately also contains an invalid transaction to some irrelevant node. The invalid transaction is there just to make sure this block will never actually make it to the legitimate blockchain, so if T trusts this transaction it may become a victim. Secure thin nodes can detect this attack by validating all the transactions in the block. This is possible but expensive in terms of shard downloads. Secure thin nodes should therefore adapt their validation cost depending on the importance of their transactions. For minor transactions, they can simply verify their own entries, but for big ones they may wish to verify all the transactions in the block. This flexibility makes secure thin nodes much more secure than legacy SPV clients.

4.2 Performance analysis

Even though a thorough performance analysis is out of the scope of this paper, in the following, we briefly discuss the storage, bandwidth, and computation requirements of secure thin nodes, and the overhead that these impose on existing full nodes.

We consider a sharding policy that splits the UTxO set in 2^k shards by looking at the first k bits of a coin hash. For our example, we assume $k = 14$, resulting into 16384 shards of 89 KiB each, considering that the UTxO set weights 1.39 GiB.

4.2.1 Secure thin nodes

Storage Secure thin nodes need to keep at least the last few blocks to resolve branches (6 MiB for 6 blocks), the list of block hashes ($427000 \times 32 \approx 13$ MiB) and the list of UTxO shard hashes ($16384 \times 32 = 512$ KiB). In total, secure thin nodes need to store at least 20 MiB. If they participate in the DHT, they additionally need to store the shards they are responsible for. To this end, they can dedicate an additional 25 MiB which enables each of them to store the two latest versions ($h = 2$) of at least 140 shards. In reality, nodes can easily save space by storing multiple versions as differences. Finally, secure thin nodes need to temporarily store shards that they download from the DHT but that they are not responsible for. To this end they can allocate a small cache (e.g., 2 MiB) as described in Section 3.3. This results in an overall storage requirement of about 47 MiB, about thirty times less than the size of the entire UTxO set.

Bandwidth When joining the network, a secure thin node initially downloads the aforementioned 45 MiB to fill its data structures and to initialize its local DHT storage. Next, for each new block, it needs to update its list of UTxO shard hashes by downloading it from full nodes, which corresponds to 512 KiB every 10 minutes. Finally, the secure thin node downloads actual shards from the DHT whenever it encounters a transaction that it wants to verify for a shard that it does not currently own. Clearly, secure thin nodes face a trade off between the number of shards they store locally either as part of the DHT or in their local cache, and the number of shards they download from other DHT nodes. A node that only wishes to verify its own transactions can minimize its storage requirements and rely on the DHT as much as possible. Conversely one that wishes to verify most transactions should dedicate more storage to have more shards readily available.

Computation When receiving a block, a secure thin node verifies the transactions relevant to its interest. Additionally, it updates the block index hash, it updates the shards it maintains based on the content of the block, and uses the updated shards to update the UTxO hash.

4.2.2 Full nodes

Storage Full nodes incur limited storage overhead. In addition to storing the entire UTxO set as in the standard Bitcoin protocol, they only need to store the list of UTxO shard hashes (512 KiB with 2^{14} shards). If they are part of the DHT, they also dedicate a few MiB of additional storage to store multiple versions of the UTxO shards they are responsible for. These can easily be stored in diff format to limit the storage requirements.

Bandwidth Full nodes that participate in the DHT incur some bandwidth overhead to honor requests of secure thin nodes, and only negligible overhead due to the additional fields in each block.

Computation Like secure thin nodes, full nodes only need to perform a few additional operations to update the block index hash, and the UTxO hash. But their cost is negligible when compared to that of block mining.

4.2.3 DHT resilience

We conclude this preliminary analysis by studying how the amount of storage dedicated to the DHT affects its resilience to node disconnections. Let us consider a scenario in which the DHT consists of 1000 secure thin nodes each dedicating 25 MiB to the DHT. As discussed above, this allows each node to store two versions of at least 140 shards. Considering a total of 16384 shards, this yields a replication factor of 8 for very minimal storage and bandwidth requirements.

5. RELATED WORK

Making the UTxO set available for queries between nodes has been proposed and discussed several times in the Bitcoin community over the past few years. Several authors have proposed to commit the UTxO set in blocks to enable faster node bootstrap [9, 13] or to also strengthen the security model of thin nodes [4, 7, 8, 10, 12, 14]; we will focus on the most advanced of these proposals.

Andrew Miller suggests [10] storing the UTxO set as leaves of a balanced Merkle tree and committing its root hash in the blockchain. Thin nodes do not need to store the tree, they can query it if it is stored by full nodes or on an external storage. To verify that a *Transaction Output* (TxO) is present in the UTxO set, a thin node starts from the leaf containing the TxO and recomputes each of it parent's hash up to the root hash. The computed root hash should match the one present in the latest block. To recompute the root hash, thin nodes only need to query for $log(n)$ tree nodes, one for each of the TxO's parent in the tree.

This solution is ideal for thin nodes, as they require minimum computation and minimum storage to verify transactions. However, full nodes need to store the entire tree, not only the leaves representing the UTxO set, which makes them much heavier. In our proposal, full nodes only pay a small additional storage cost.

Peter Todd has a more radical approach [15] in mind when it comes to dealing with the growth of the UTxO. The TxO set, both spent and unspent, is stored in insertion ordered Merkle trees (named Merkle Mountain Range, MMR, by the author) which can easily be pruned and restored when needed. Pruning the MMR enables a lower storage requirement, and can be done when a subtree of the MMR is committed in the blockchain. The restoration of a subtree is needed when a transaction spends an old TxO present in a pruned subtree. In such a case, the transaction must be accompanied with a proof that the TxO is present in the MMR, such as the list of the TxO parents in the tree. This proof can be stored by the owners of the spendable coins, rather than by the verifying nodes. At the extreme, no node needs to keep the UTxO set if all the transactions include a proof of validity

Peter Todd's proposal imposes a profound change to the Bitcoin protocol by adding proofs of validity to the transactions themselves, thus shrinking the storage requirements, but drastically increasing the bandwidth consumption.

The scalability of Bitcoin has been a growing concern as adoption kept increasing. In this section, we have presented several proposals that deal with storage scalability, but more scalability challenges have been identified in the literature [1, 2, 3]. In particular, Bitcoin-NG [6] focuses on the throughput scalability and reach a near optimal number of transaction per second.

6. CONCLUSIONS

We have presented an adaptive distributed storage solution for Bitcoin that improves the security model of thin nodes, such as mobile devices, by enabling them to *locally* verify transactions in blocks as they are being created. Our secure thin nodes can verify transactions by querying a DHT for shards of the set of unspent transaction outputs (UTxO) as well as for blocks when searching in the blockchain history. Secure thin nodes can define the maximum storage space they wish to allocate for Bitcoin, inducing a user-defined trade-off between storage requirement and bandwidth consumption.

For the future, we plan to evaluate our solution, as well as further tuning the UTxO sharding policy to reduce the number of UTxO shards needed to verify a block.

Acknowledgments

This work has been partially funded by the Region of Brittany, France, by the Doctoral school of the University of Brittany Loire (UBL), by the French National Research Agency (ANR) project *SocioPlug* under contract ANR-13-INFR-0003 (http://socioplug.univ-nantes.fr) and by the Dutch national research program COMMIT/ (a public-private research community).

7. REFERENCES

[1] B. Bishop. Review of Bitcoin Scaling Proposals. In *Scaling Bitcoin Workshop Phase 1*, Sept. 2015.

[2] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten. SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. In *2015 IEEE Symposium on Security and Privacy*, pages 104–121, May 2015.

[3] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, and E. Gün. On scaling decentralized blockchains. In *Proc. 3rd Workshop on Bitcoin and Blockchain Research*, 2016.

[4] DiThi, Jan. 2012. https://en.bitcoin.it/wiki/User:DiThi/MTUT.

[5] V. Durham. Namecoin, 2011. https://www.namecoin.info.

[6] I. Eyal, A. E. Gencer, E. G. Sirer, and R. V. Renesse. Bitcoin-ng: A scalable blockchain protocol. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, 2016.

[7] M. Friedenbach, Mar. 2014. https://github.com/bitcoin/bitcoin/pull/3977.

[8] G. Maxwell, June 2011. https://bitcointalk.org/index.php?topic=21995.0.

[9] B. McElrath, Oct. 2015. https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2015-October/011638.html.

[10] A. Miller. Storing UTXOs in a Balanced Merkle Tree (zero-trust nodes with O(1)-storage), Aug. 2012. https://bitcointalk.org/index.php?topic=101734.0.

[11] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[12] A. Reiner, June 2012. https://bitcointalk.org/index.php?topic=88208.0.

[13] R. K. Svendsen, Sept. 2015. https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2015-September/011065.html.

[14] TierNolan, May 2015. https://bitcointalk.org/index.php?topic=1048021.0.

[15] P. Todd. Making UTXO Set Growth Irrelevant With Low-Latency Delayed TXO Commitments, May 2016. http://petertodd.org/2016/delayed-txo-commitments.

[16] G. Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 2014.