



**School of Mechanical and Manufacturing Engineering**  
**Faculty of Engineering**  
**The University of New South Wales**

# **A Storage Flexible Blockchain**

by

**Benjamin Lee**

Thesis submitted as a requirement for the degree of  
Bachelor of Engineering in Mechatronic Engineering

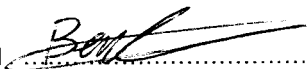
Submitted: October 2018  
Supervisor: Dr. Mark Whitty

Student ID: z3463414  
Co-Supervisor: Dr. Salil Kanhere

#### ORIGINALITY STATEMENT

'I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, or substantial proportions of material which have been accepted for the award of any other degree or diploma at UNSW or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by others, with whom I have worked at UNSW or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic expression is acknowledged.'

Signed



Date

21/10/2018

# Abstract

The use of blockchain in applications outside of cryptocurrencies is increasing due to its desirable features. However, a problem that may be encountered in high volume applications of blockchain may be that the large storage required may be a barrier for long term use due to the large amount of data being processed and stored. A new method of processing blockchain data is proposed where participants in the blockchain are given more flexibility in how their data is managed, with the possibility of reducing the storage footprint of the blockchain by deleting created transactions or aggregating multiple transactions into a single transaction. These actions can be performed by either the creators or the processors of the data, further increasing the flexibility provided in the system. Different configurations of this system are tested to benchmark against a more traditional implementation of blockchain to determine the feasibility of reducing the storage requirements through providing storage flexibility in a blockchain.

# Contents

<b>List of Figures</b>	<b>iv</b>
<b>List of Tables</b>	<b>v</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Literature Review</b>	<b>3</b>
2.1 Context . . . . .	3
2.2 Blockchain Scaling . . . . .	4
2.3 Database Storage . . . . .	8
<b>3 Methodology</b>	<b>9</b>
3.1 General Blockchain Implementation . . . . .	9
3.2 Extending the Blockchain for Storage Flexibility . . . . .	11
3.3 Miner Initiated Reduction . . . . .	13
3.3.1 Temporary Transactions . . . . .	13
3.3.2 Miner Summarisable Transactions . . . . .	13
3.4 User Initiated Reduction . . . . .	14
3.4.1 Generator Verifiers . . . . .	15
3.4.2 Remove Transactions . . . . .	17
3.4.3 Summarise Transactions . . . . .	18
3.5 Cleaning Period . . . . .	18
3.6 Correctness . . . . .	20
3.7 Performance Indicators . . . . .	20
<b>4 Evaluation</b>	<b>22</b>
4.1 Results . . . . .	23
4.1.1 Temporary Transactions . . . . .	23
4.1.2 Miner Summarisable Transactions . . . . .	26
4.1.3 Remove Transactions . . . . .	27
4.1.4 Summarise Transactions . . . . .	29
4.1.5 Testing On Large Blockchain . . . . .	31

4.2 Discussion . . . . .	33
<b>5 Conclusion</b>	<b>38</b>
<b>References</b>	<b>40</b>
<b>6 References</b>	<b>40</b>
<b>Appendices</b>	<b>44</b>
Appendix A Pseudocode for summarising a list of transactions	44
Appendix B Pseudocode for verifying the generator verifier	44
Appendix C The miner testing code	44

## List of Figures

1 The basic chaining structure in a blockchain . . . . .	1
2 How Merkle pruning works in Bitcoin . . . . .	5
3 Fields in a blockchain transaction. . . . .	10
4 An example of how summary transactions work. . . . .	12
5 The fields populated in a permanent transaction. . . . .	13
6 The fields populated in a temporary transaction. . . . .	13
7 The fields populated in remove and summarise transactions. . . . .	17
8 The impact of transaction location when removing from the blockchain. . .	23
9 The storage footprint and processing time for temporary transactions. . . .	24
10 The ratio of processing time for temporary transactions to permanent trans- actions. . . . .	25
11 Peak RAM usage and thread count for temporary transactions. . . . .	25
12 The storage footprint and processing time for miner summarisable trans- actions. . . . .	26
13 The ratio of processing time for miner summarisable transactions to per- manent transactions. . . . .	27
14 Peak RAM usage and thread count for miner summarisable transactions. .	28
15 The storage footprint and processing time for user remove transactions. . .	29

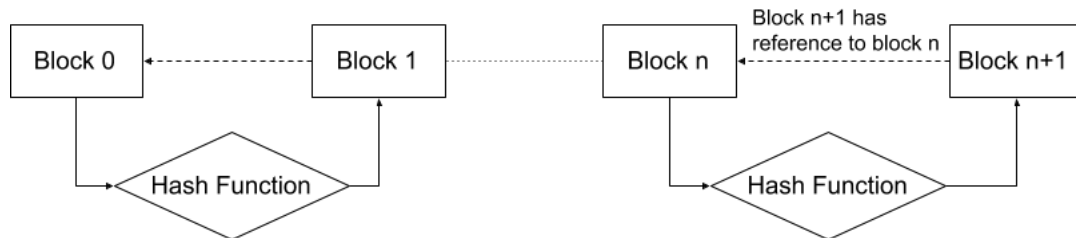
16	The ratio of processing time for user remove transactions to permanent transactions. . . . .	30
17	Peak RAM usage and thread count for user remove transactions. . . . .	31
18	The storage footprint and processing time for user summarise transactions.	32
19	The ratio of processing time for user summarise transactions to permanent transactions. . . . .	33
20	Peak RAM usage and thread count for user summarise transactions. . . . .	34
21	Number of summarise transactions created for different cleaning intervals. .	35
22	The effect of changing the cleaning interval and number of blocks on the storage requirements of the blockchain. . . . .	36

## List of Tables

1	Time and cost required to achieve a 1GB storage saving for temporary transactions . . . . .	34
2	Time and cost required to achieve a 1GB storage saving for miner summarisable transactions . . . . .	35
3	Time and cost required to achieve a 1GB storage saving for remove transactions . . . . .	36
4	Time and cost required to achieve a 1GB storage saving for user summarise transactions . . . . .	37

# 1 Introduction

Blockchain as a technology is most well known as one of the core parts to Bitcoin and other similar cryptocurrencies. A blockchain allows participants in the system (called nodes) to send data amongst themselves by storing information in transactions. Typically, transactions contain additional pieces of information such as timestamps and the public key of the sending node, and are digitally signed by the nodes transmitting the information with a public/private key pair held by each node to prevent forgery. Transactions are aggregated by special nodes, miners, in data structures called blocks. Miners in the system keep a list of all the past blocks that have been created as a chain (hence blockchain). This allows miners to be able to follow the blocks from the latest to the original and verify each individual block. In addition to storing transactions, blocks also store the merkle tree created from all of the transactions and the cryptographic hash of the last block created [2]. By doing so, miners can repudiate any edits to past transactions stored in blocks and prevents replacement of past blocks in the blockchain. See Figure 1 for a simple diagram on how the chaining mechanism works. From the latest block created, a miner can follow the chain to the first block created, also known as the genesis block, and verify that each block has not been tampered with.



**Figure 1: The basic chaining structure in a blockchain**

The features of a blockchain that make it useful for cryptocurrency applications such as immutability, security and privacy make blockchain attractive to applications outside of cryptocurrencies. Crosby et al. have identified various industries that could benefit from incorporating blockchain into their existing workflow, including the Internet of Things (IoT), healthcare, finance and law [3]. This can be observed in real world situations already with Malta proposing regulatory frameworks based on blockchain or the NSW employing the use of blockchain to underpin their rollout of digital driver's licences and potentially other applications such as birth certificates and property titles [4] [5] [6]. Ad-

ditionally, Alibaba Cloud, a cloud computing company that offers services, including high volume ones such as content delivery networks and domain name resolution have started trialling a new "blockchain as a service" which could see even high volume applications employ the use of blockchain due to their desirable features [7].

However, one limitation of blockchain in its current form is that for larger scale applications or those that require processing large volumes of transactions, the storage footprint of the blockchain becomes infeasible to manage. Most current research into scaling blockchain focuses on increasing the throughput but there is not a great deal of research into scaling the storage because storage is cheap, with one gigabyte of storage now costing less than \$0.05 [8].

In the current, prevailing implementation of blockchains, transactions are generally stored indefinitely on the blockchain with no method of deleting unnecessary data. Some implementations, such as Bitcoin, have automated methods of deleting old data but this depends on there being enough new transactions to bury the old transaction [1]. This paper attempts to address the challenge of large sized blockchains by introducing a method of deleting past transactions while maintaining the desirable features present in a blockchain. The idea of storage flexibility is presented, where miners and nodes in a blockchain are given different options as to how data is stored and processed with more control over whether data is stored or removed than is currently provisioned in traditional blockchain implementations.

Chapter 3 describes how the blockchain in this paper functions and how the system provides extra flexibility to both users and miners. The metrics used to measure the performance of this system is also described in this chapter. Chapter 4.1 shows the results of simulating the implemented storage flexible blockchain and analyses the results of the simulations. A discussion of these results and of the overall blockchain is provided in this chapter.



## 2 Literature Review

### 2.1 Context

Currently, the size of the two largest blockchains used in cryptocurrencies, Bitcoin and Ethereum is 198.8 GB and 606.3 GB respectively [9]. These sizes are manageable with readily available consumer storage systems. Other applications for blockchain outside of cryptocurrency however, could possibly see blockchain sizes that far exceed those in cryptocurrencies and thus require some method of dealing with a large storage footprint that is not addressed by blockchains in their current implementations. For example, IoT service providers could utilise blockchain to keep some sort of immutable ledger for data that is processed by their internet connected devices [3].

IHS Markit, an information services company, estimates that there were 27 billion connected IoT devices in 2017 and that this figure will increase to 125 billion by 2030 [10]. If even a small percentage of these devices were to use the blockchain to exchange data, it may be possible that the amount of transactions generated could subsequently overwhelm the storage capacity currently used in blockchain. Deletion of transactions may be provisioned but the volume of data generated potentially could outpace the deletion using current removal methods.

Additionally, Cisco estimates that by 2021, IoT devices will create 847 ZB (one trillion Gigabytes) of data and of that data, store 8.47 ZB [11]. Again, this presents a problem in that if some IoT services decided to utilise blockchain to provide an immutable history of processing of data stored or generated, it could quickly overwhelm storage solutions used.

Even without considering an application like IoT, Poon and Dryja demonstrate that the current framework for cryptocurrencies like Bitcoin is not scalable [12]. Bitcoin can handle around 7 transactions per second compared to a major payment network like Visa achieving around 1700 transactions per second on average, although the peak can reach 56,000 transactions per second [13] [14]. Using Poon and Dryja's assumptions of 300 byte transactions and unlimited transactions per block, reaching Visa levels of transac-

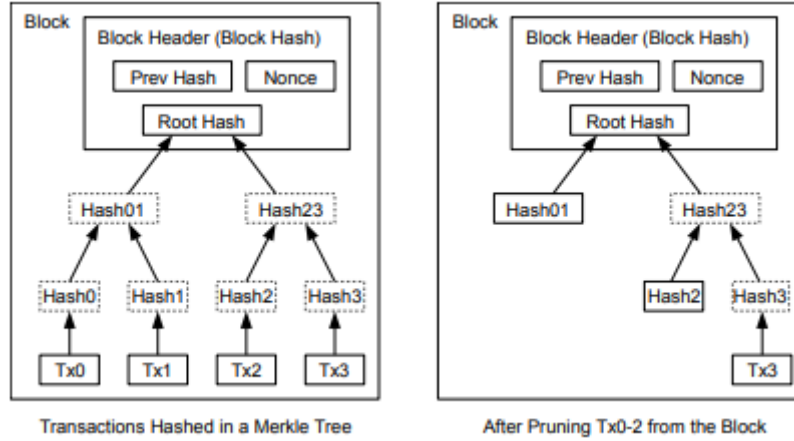
tion processing would create 306MB blocks every 10 minutes totalling 18TB per year and at peak could create 10GB blocks every 10 minutes totalling 529TB per year. Naturally, Bitcoin would not reach transaction rates like Visa's as the adoption of Bitcoin as a payment system is not nearly as significant as Visa but it demonstrates that storage of the blockchain quickly becomes an issue if high volumes of transactions were considered.

General Data Protection Regulation (GDPR), the regulation recently introduced by GDPR defines the rights that European Union residents can expect regarding their data and how it is used, protected and stored [15]. The regulation states that one of the rights European residents are afforded is "the right to be forgotten" where they can request their personal data to be deleted and no longer processed. In a blockchain, transactions and their data could be considered personal data especially if the public key can be associated with personally identifiable information. As a result, most blockchains in their current implementations would not be GDPR compliant as they do not allow for personal data erasure on request but instead only delete old transactions for disk space reclamation [1]. The flexible blockchain implemented here would address this issue as it allows for users in the blockchain to request for deletion of transactions tied to their public key and as a result, erase their personal data on the blockchain.

## 2.2 Blockchain Scaling

Blockchain was first formally introduced by Nakamoto in their original Bitcoin paper. [1] In it, Nakamoto defines the concept of a blockchain as the technology that backs the decentralised cryptocurrency known as Bitcoin. The properties inherent to a blockchain are proved in Nakamoto's paper. Nakamoto briefly mentions disk space reclamation in their paper by pruning old transactions, deleting them and only keeping the Merkle Tree hashes of transactions. This way, the integrity of other transactions can be maintained by allowing nodes to verify the still existing transactions to their hashes from the root hash to all leaf nodes. Fig 2 from Nakamoto's paper demonstrates demonstrates how Bitcoin's blockchain safely removes transactions [1]. However, this method only removes transactions that, tied to a coin (a unit of currency in Bitcoin) have become sufficiently old in the blockchain. For applications that do not require the use of coins, this pruning method may not be feasible as no transactions are tied to a coin unless all

transactions that reach a certain age are pruned. Additionally, some applications may require permanence of certain transactions and the Bitcoin method of pruning would not fit the requirements in these situations. A blockchain that has multiple transaction types would address both of these points as old transactions can be safely removed from the blockchain while keeping those that should not be deleted.



**Figure 2: How Merkle pruning works in Bitcoin [1].**

Szyldo presents a method to efficiently traverse Merkle Trees [16]. He proves in his paper that there is no method that is more efficient in traversing Merkle Trees than in logarithmic space and time. Since Bitcoin takes advantage of Merkle Trees in its transaction pruning process, Szyldo’s method can be applied in the Merkle Tree traversal during pruning. Since one of the requirements for the blockchain to be implemented is removal of specific transactions, Szyldo’s results can be used to provide this service in an efficient manner.

Kroman et al. investigate a method to address the issue of scaling blockchain by parameterisation of block sizes in the system [17]. They concede however, that such a measure would not solve the problem entirely and that scaling would potentially require redesign of systems and rethinking how it is approached. One such aspect of the blockchain that could be redesigned to aid scalability is in its storage approach; by implementing a system with more than one transaction type, flexibility is provided in how transactions are handled by the system.

Another method of scaling blockchain is explored by Dennis et al. where the issue of

having a fast growing blockchain is solved by using a fixed size blockchain [18]. In their "rolling blockchain", a pre-set age is defined whereby all data older than that age is automatically removed from the blockchain, providing scalability since the blockchain is prevented from growing beyond a certain size. This solution is a more extreme version of the blockchain pruning that is performed by Bitcoin as it no longer even stores older blocks beyond a certain age. The benefit of this method is that the system is able to keep the blockchain and thus storage footprint at a certain size magnitude without compromising the security of the system. However, this would not be applicable for systems where data should be stored long term as it indiscriminately erases all data after a certain time.

Raman and Varshney introduce a method of scaling blockchain storage without sacrificing the potential size of the blockchain by distributing the storage of block transactions over the miners in the system [19]. Symmetric encryption is used to distribute the transaction blocks in a confidential manner. The secret key used in the encryption is split amongst the miners that the transaction blocks are distributed to, ensuring that no single miner is able to decrypt the transaction data by themselves. However, this method of storage reduction comes with some drawbacks as well. In order to audit any transaction on the blockchain, the system requires that there is a minimum number of miners that are storing the transaction blocks to collude in order to recreate the secret key and decrypt the original transaction from its blocks. This opens up the system to denial of service attacks as an adversary could make it impossible to decrypt transactions just by targeting enough miners storing particular transactions. Additionally, the process of restoring a transaction from its constituent blocks introduces additional computational overhead because of the secret key recreation, requiring communicating with several miners.

Bruce proposes another method of scaling blockchain by having a fixed size blockchain [20]. Instead of deleting blocks that have lived past a predefined age, the blockchain size is set to limit the number of blocks allowed, with older blocks being removed from the blockchain as newer ones are appended if the size is greater than the pre-set limit. Compared to the temporal blockchain in [18], this blockchain could likely be easier to maintain as the blockchain size is always maintained whereas the temporal blockchain could possibly see an influx of blocks being appended in quick succession leading to a larger size blockchain

until the blocks are aged out. Bruce’s method runs into the same limitations as the temporal blockchain where blocks are purged from the blockchain without regard to whether certain data should be kept long term or not.

Ateniese et al. present a blockchain framework that has the ability to edit block contents in a blockchain [21]. Their framework uses chameleon hash functions which allow efficient finding of hash collisions, given extra information known as a trapdoor. By being able to find a collision, nodes in a blockchain are able to change the contents of a particular block while still satisfying the integrity of the blockchain. Due to the nature of cryptographic hash functions used in blockchains such as Bitcoin, finding a collision is supposed to be computationally expensive and so this would not be possible without the use of the aforementioned chameleon hash functions. However, the method presented by Ateniese requires the modified blockchain to be sent to all nodes in the system with each node replacing their currently stored blockchain with the received one. This would most likely be infeasible for large scale applications if many nodes request editing of blocks sequentially as it would require repeatedly sending a new copy of the blockchain upon every edit. Additionally, the editing of the blockchain is dependent on a “central authority” which defeats the decentralisation aspect of most blockchains. Ideally, any node would be able to request a change on a transaction that was generated by them and the edit would be able to be performed on the stored blockchain and without the need of a central authority. The redactable blockchain demonstrated could potentially help reduce storage requirements as transactions could be removed by editing blocks but this is not an efficient process as mentioned above. Bennett et al. also propose the idea of using a chameleon hash function to open blocks for editing without violating the integrity of the chain although it is unclear whether the idea is the same as that of Ateniese’s [22].

A real world example of editing of the blockchain can be observed in the Ethereum blockchain [23] [24]. Due to an attacker exploiting how the Ethereum blockchain was implemented, the blockchain’s currency (ETH) was able to be transferred from victim smart contracts into one the attacker controlled with over \$60 million stolen [25]. This led to the Ethereum blockchain being split at the point before the attacker began stealing ETH and regular Ethereum activity resuming from that point in a process known as forking.

This led to all of the transactions from all users after that point to be lost so this process of blockchain modification is obviously not feasible and was only done as there was no other way to reverse the transactions.

Dorri et al. present a storage optimised blockchain in their paper that allows the system to reduce the storage footprint in a flexible manner [26]. However, one of the limitations of the system implemented was that there was a performance bottleneck such that a large volume of transactions cannot be handled efficiently. This becomes an issue at larger scales and thus the blockchain would be infeasible to use but the paper addresses the issue of storage footprint such that transactions can be both preserved and erased based on system needs rather than age of data. The blockchain implemented in this paper builds upon Dorri's work by improving the efficiency at which data is handled, potentially making such a system viable for newer, large scale blockchains.

## 2.3 Database Storage

Nayak et al. discuss the two major types of database management systems in their article, namely NoSQL and Relational Databases, more commonly known as SQL databases [27]. These two types of database are compared for different use cases with advantages and disadvantages of both being explored. They conclude that NoSQL databases tend to be easily scalable, faster, more efficient and flexible at the cost of higher maintenance, the loss of a standard language and interface and immaturity in the technology. Since the aim of the blockchain being implemented is flexibility and this requires fast read/write speeds, the research presented in this article helps determine that NoSQL would be the best choice for the database type backing the blockchain. Several NoSQL database implementations are touched upon briefly but there are no statistics that help determine which database in particular would best suit the needs of the blockchain.

Alex provides database benchmarks for certain NoSQL Key-Value stores [28]. Although the databases compared are implemented in Java, their general performance in other language implementations can be inferred from Alex's benchmarking. Although multiple statistics are presented in the benchmarks, there are three that should be considered the most important; these are disk use and random access (read/write). This is due to

the fact that block hashes will be used as keys in the database which are most likely not sequential in value between blocks. In terms of access speed, LMDB is the fastest database available but the storage overhead involved with using it is too high; with up to 65% of disk use as overhead, this would be unacceptable for our application as the aim is to reduce the storage footprint. The databases with the least storage overhead are leveldb and RocksDB with overheads of just under 3% which is acceptable. These two database implementations are also sufficiently fast, with random read/write speeds not significantly less than those of LMDB. The sequential read/write speeds for leveldb and RocksDB are slow in comparison with other databases but because there will likely be very little sequential access from the blockchain, this is not a large factor. A comparison between RocksDB and leveldb is also made in Alex’s benchmarks, with RocksDB shown to outperform leveldb in disk access. Facebook’s benchmark comparisons between RocksDB and leveldb draws the same conclusion that RocksDB is superior to leveldb although there may be some bias as RocksDB is created by Facebook [29]. Both RocksDB and leveldb provide single process, multi-threaded access to a database without the need for additional synchronisation primitives in code [30] [31]. This is ideal as the implementation will likely require multiple threads to handle all the blockchain operations in the implementation in order to provide storage flexibility without compromising performance and the provided concurrency control will make the implementation code simpler.

## 3 Methodology

### 3.1 General Blockchain Implementation

The overarching framework is still heavily based on the blockchain presented by Nakamoto [1]. Generally, a blockchain revolves around four entities: transactions, blocks, nodes and miners (a subset of nodes). A brief description of these entities is given here.

A transaction contains various fields; these are shown in Fig 3. The *TxID* and *PrevTxID* fields represent the transaction ID and previous transaction ID where the ID of a transaction is the cryptographic hash of the other fields in the transaction. The inputs and outputs to a transaction are stored in the *Input* and *Output* fields; for Bitcoin, an input would be a payment to a user’s wallet and the output would be where the recipient wallet

sends the coins received. For other applications, the input and outputs would be use case dependent. The *PubKey* field contains the public key associated with a user; this value should be unique per user. The *Sign* field is the associated digital signature generated based off the transaction fields outside of the ID. The signature, along with the public key, is used to verify the authenticity of transactions sent.

TxID	PrevTxID	Input	Output	PubKey	Sign	Time
------	----------	-------	--------	--------	------	------

**Figure 3: Fields in a blockchain transaction.**

A block is the data structure that stores multiple transactions. The general structure of a block is shown in the left part of Fig 2. The Merkle Tree is generated by recursively hashing the block's transactions, represented as leaves in the tree. A block has a defined size in terms of the number of transactions that can be stored; once there are enough transactions that have been sent, a new block is created and appended to the blockchain. The *Block Header* is the hash created from the Merkle root, the previous block hash and a nonce. The nonce is used in the consensus protocol for blockchain applications; as the consensus protocol will not be implemented in this paper, it is safe to ignore and exclude the nonce from the block header. This does not compromise the block hash generated as the merkle root hash should almost always be different for any unique set of generated transactions. Naturally, these components are also stored in the block as well so that the block hash can be verified and so that a node can follow the chain to the genesis block.

A node is a participant in the blockchain. Nodes create transactions and have their own public/private key pairs which they use to sign transactions that they create. The created transactions are broadcast to all other nodes in the network and can then be processed by miners.

Miners are special nodes that process received transactions, add them to blocks and then append blocks to the existing blockchain in a process called mining. Part of the processing of transactions requires miners to verify the authenticity of transactions received by checking the digital signature in the transaction using the public key of the sending node.



Mining blocks involves a special process called the consensus protocol. In this protocol, miners are required to solve some problem that is computationally difficult but easy to verify. This prevents rogue nodes from creating fraudulent transactions and adding them to a block as honest nodes would be able to check if the problem was successfully solved and if not, disregard the block that was added. As the aim of this paper is to investigate a storage flexible blockchain implementation, this functionality is not implemented and it is assumed that the participating nodes are honest, although this will not necessarily be true in real world applications.

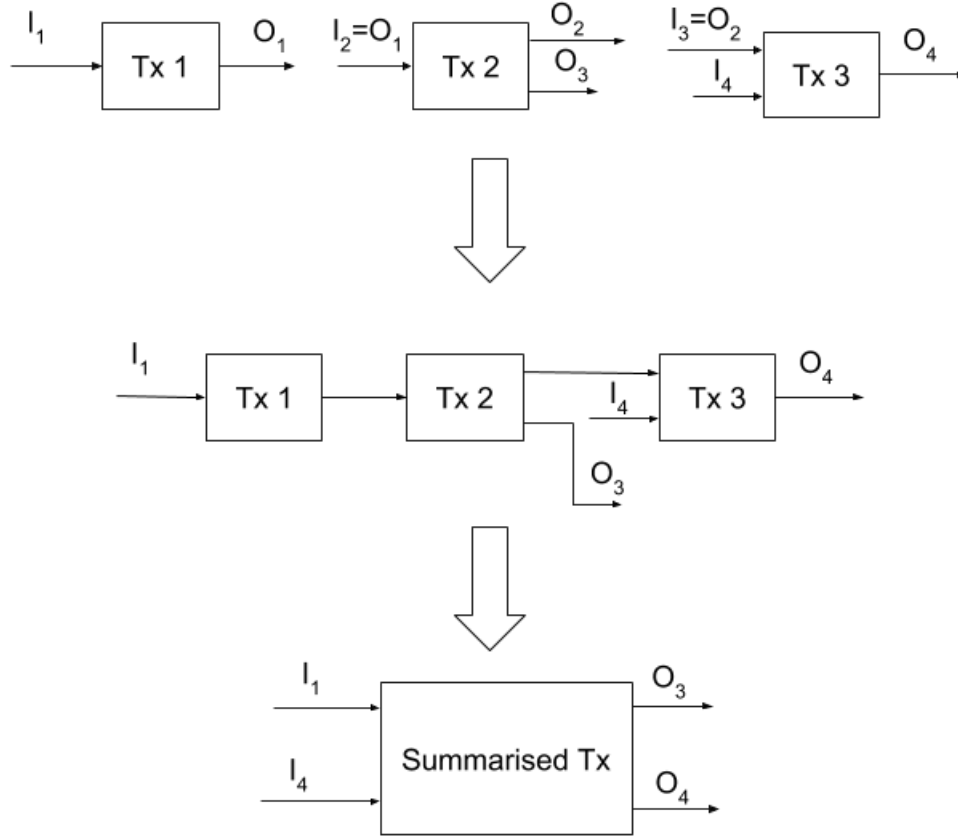
## 3.2 Extending the Blockchain for Storage Flexibility

In order to reduce the storage footprint of a blockchain, two reduction mechanisms are proposed: summarising and removing. Removing, as the name implies, deletes a transaction from the blockchain. This is performed in a similar manner to how Nakamoto describes in their Bitcoin paper but there is greater control over when transactions are deleted in this implementation. Summarising transactions reduces a set of transactions into a single transaction by combining inputs and outputs. The combined transactions are subsequently removed from the blockchain. These two reduction methods can be invoked by either the miner or by a node; however, a node can only remove or summarise transactions that they have created.

In order to reduce the storage requirements for transactions, the public key field in all transactions is replaced by a public key hash. When a user first connects to the system, they broadcast their public key to the miners. The miners can then hash the public key and store it in a dictionary for future use. When miners receive a transaction, they can retrieve the public key hash of the user from the *PubKey* field, which they can use to index into their dictionary to retrieve the original public key. This reduces the amount of bytes in each transaction and consequently, the amount of storage required overall in the blockchain.

Summarising transactions is a process to reduce the storage footprint of the blockchain by taking multiple transactions and aggregating their inputs and outputs into one trans-

action. There is a storage reduction as long as there is more than one transaction being summarised because the process generates one transaction on the blockchain every time a set of transactions become summarised. Figure 4 outlines how transactions become summarised into a single transaction. Pseudocode to create a summary transaction given a list of transactions can be found in Appendix A.



**Figure 4: An example of how summary transactions work.**

A concept of transaction types is introduced so the miner is able to determine how to correctly process the transactions received. A new field is added to transactions generated to indicate their type which the miner will check. Transactions that should always remain on the blockchain, similar to how the current blockchain stores transactions, will have a *permanent* type. Additionally, four transaction types are introduced. Two of these will indicate miner initiated reduction and the other two indicate node/user initiated reduction. In this storage flexible blockchain, new fields are added to transactions to store additional processing information that miners will require. The types and the extra fields are discussed in further detail below. For permanent transactions, a new field is the

aforementioned type field, with the structure shown in Figure 5.

TxID	PrevTxID	Input	Output	PubKey	Sign	Time	Type
------	----------	-------	--------	--------	------	------	------

**Figure 5: The fields populated in a permanent transaction.**

### 3.3 Miner Initiated Reduction

The two types that indicate a transaction is eligible to be processed later by a miner will be called *temporary* and *summarisable* to represent transactions that will be removed and summarised, respectively.

#### 3.3.1 Temporary Transactions

As the name suggests, temporary transactions are those that exist on the blockchain for a period of time, after which they will be removed by the miner. An additional field, the time to live (*TTL*), is added to transactions so that miners will know how long to keep the transaction for. The transaction ids of all the transactions that are marked as temporary are kept in main memory by the miner so that the miner can track what transactions need to be removed from the blockchain and the appropriate time to remove them. The *TTL* field will only ever exist and be processed by the miner if the transaction type is set to temporary. The fields in a temporary transaction in Figure 6 show the fields that will be populated in a temporary transaction.

TxID	PrevTxID	Input	Output	PubKey	Sign	Time	Type	TTL
------	----------	-------	--------	--------	------	------	------	-----

**Figure 6: The fields populated in a temporary transaction.**

#### 3.3.2 Miner Summarisable Transactions

The other type of transactions that can be removed by the miner are miner summarisable transactions. In this type of storage reclamation, a set of transactions are summarised into one transaction, and all of the transactions that were summarised are then removed

from the blockchain. As summarisable transactions need no further information for the miner to process, they take on the same structure as permanent transactions, shown in Figure 5. When a miner summarises transactions, they summarise all transactions that are marked as miner summarisable, regardless of the creator of the transaction.

### 3.4 User Initiated Reduction

The system described in this paper also provides users (non miner nodes) flexibility in how the data that they create is stored. Similar to the miner initiated reductions, users can also cause their sent transactions to be either removed from the blockchain or summarised. However, this process requires that the user is able to prove that they are the original creators of the transaction that they want to operate on otherwise a malicious actor would be able to indiscriminately cause any transaction to be deleted from the blockchain. As a result, the concept of a generator verifier is introduced to address this. Chapter 3.4.1 describes the generator verifier and its use in further detail.

All user initiated reductions cause miners to perform some operation on existing transactions on the blockchain. In order for miners to know which transactions the operations are to be performed on, the sending node must include something to identify the transactions. The transaction id is the easiest way to indicate which transactions the miner should operate on but this information also needs to be included in the calculation of the transaction's digital signature so that the transaction ids cannot be tampered with. If each transaction id were to be included in the digital signature, it may be computationally expensive for a miner to verify the digital signature if a user were to include a large set of transaction ids as the miner would have to take into account every transaction id when verifying the digital signature whilst still performing other tasks such as receiving transactions and creating new blocks. Instead, the sending node will create a merkle tree from the transaction ids of which the tree root can then be included in the calculation of the digital signature. Using the merkle tree root in the calculation of the transaction digital signature instead of a list of transaction ids allows for more efficient signature verification while still providing resilience against tampering as all the transaction ids sent by the user can also be verified by recursively hashing up to the merkle root which is used

in the digital signature. The merkle root is thus added as a field to all user reduction transactions from which the miner can recover all the transaction ids the user wishes to operate on.

### 3.4.1 Generator Verifiers

A generator verifier, as the name suggests, is a piece of information attached to transactions that will allow miners to verify that a node was the creator of that transaction. This means that it needs to contain a piece of information (from here referred to as a generator verifier secret) that only the creator knows, for example, a password. Additionally, the generator verifier must be unique for every transaction to prevent replay attacks from occurring. This can be easily achieved by adding a unique piece of information to the creation of the generator verifier. An example of this is the transaction id used in all transactions. Although it may be possible for two transactions to have the same transaction id, it would require on average  $\sqrt{n}$  values, where  $n$  is the number of possible outputs, to obtain a greater than 50% chance of getting two outputs the same [32] (assuming equally probable outputs). As the hash function used in the generation of transaction ids (SHA256), has  $2^{256}$  possible outputs, then to achieve a greater than 50% probability of getting a collision in transaction ids would require  $2^{128}$  or approximately  $10^{38}$  outputs. Reducing that probability also decreases the number of outputs required but even to achieve a  $10^{-120}\%$  probability of a collision would require approximately  $10^{32}$  outputs. These values required for the latter probability would take a system that generates a trillion transactions a second more than a trillion years to achieve and thus it can be safely assumed that transaction ids are unique.

However, the generator verifier included in transactions must be changed in some way such that the generator verifier secret of users are not revealed. A hash function is used to do this; the benefits of using a hash function are that the output of the hash is not reversible so it would be computationally intractable for an adversary to determine the generator verifier secret if given a generator verifier and also because the output of a hash function is fixed length so that the generator verifier will not take up a large amount of storage if a user decides to have a large sized generator verifier secret. Thus a generator verifier is created by the following  $verifier = H(secret + transaction_id)$ , where  $H$  is the

cryptographic hash function used.

When a node generates a transaction, the transaction id is calculated by the object constructor. The generator verifier value is used to encrypt the transaction id and the encrypted value is stored in the transaction in a new field; this value is associated with every transaction created so the new field is added to all transaction types. When a user wishes to remove or summarise existing transactions on the blockchain, they generate the merkle tree from the transaction ids. From the list of transaction ids, the user also generates a list of generator verifiers using the method mentioned above, where the transaction id used in the calculation is each id contained in the merkle tree. This is sent to the miners in the remove or summarise transaction. When a miner receives the transaction from the user, they can retrieve the transaction ids from the merkle tree and for each transaction on the blockchain corresponding to the retrieved transaction ids, the miner can verify ownership of the transaction by decrypting the encrypted value in the retrieved transaction using the corresponding generator verifier sent in the received remove or summarise transaction. If the decrypted value matches the transaction id, then that generator verifier is considered verified. All of the generator verifiers must return true otherwise the miner can discard the entire remove or summarise transaction received. Only upon verifying the entire list of transactions will be received transaction be added to the pool of transactions to be mined to the blockchain. See Appendix B for pseudocode of the generator verifier creation and verification processes. This benefit of using the verifier as an encryption key is that the verification of a user's ownership of a transaction depends only on the generator verifier allowing a user to change their public/private key pair in this system and still effect changes on their past transactions as long as they keep track of their generator verifier secret.

This method of verification is safe from adversaries as long as the encryption method is resilient against known plaintext attacks (for example, AES). An adversary must be able to either guess the generator verifier secret or guess every generator verifier for a given set of transactions in order to impersonate a user wishing to cause changes on past transactions. Assuming that the cryptographic hash function used provides preimage resistance, it would be infeasible for an adversary to guess every generator verifier; guessing

the generator verifier secret would likely be easier. As such, it is recommended that when creating the generator verifier secret, the user treats it like a password and uses a hard to guess sequence. To provide additional resilience against guessing the generator verifier secret, the system allows the secret to change because the generator verifier is only dependent on the secret which need not be the same for different transactions. In this case, an adversary would have to guess the secret for every generator verifier for a given set of transactions which would be infeasible if a strong secret were used for every verifier created. The only additional cost for the user is that they would either have to store the secret for every transaction created or have some other method of tracking the state of the secret as the transactions are created (for example, applying a fixed transformation for each transaction).

### 3.4.2 Remove Transactions

When a user wishes to remove certain transactions from the blockchain, they recreate, for each transaction, the associated generator verifier values described in 3.4.1 and create a merkle tree from the transaction ids. These list of generator verifier values and the transaction id merkle tree are placed in a remove transaction which is sent to the miner. Figure 7 shows the structure of a remove transaction which will be processed by the miner.

TxID	PrevTxID	Input	Output	PubKey	Sign	Time	M.R	G.V.L
------	----------	-------	--------	--------	------	------	-----	-------

**Figure 7: The fields populated in remove and summarise transactions. M.R and G.V.L are the merkle root and generator verifier list objects, respectively.**

When a miner receives a remove transaction, it unpacks the transaction ids from the merkle root by performing a traversal through the tree. The search returns a list of the transactions in the order they were inserted into the merkle tree; by knowing the index of a particular transaction id, the associated generator verifier for that transaction can be retrieved in constant time by indexing into the generator verifier list at that index. During the next cleaning period after receiving the remove transaction, the miner searches through the blockchain for every transaction object listed in the merkle tree. When a transaction is found, the miner verifies the given generator verifier using the process described in Chapter 3.4.1 and Appendix B. If at any point the verification fails, the

remove transaction is discarded. If all of the generator verifiers are successfully verified, then all the transaction ids are queued in the miner for removal during the next cleaning period.

### 3.4.3 Summarise Transactions

A user wishing to summarise transactions instead of removing transactions largely follows the same process as creating a remove transaction; the merkle tree is constructed from a list of transaction ids and the associated generator verifiers are calculated from the transactions the user wishes to summarise. As such, the structure for a user summarise transaction is the same as that of a user remove transaction, shown in Figure 7. However, the difference is that the user also determines the inputs and outputs to the transaction, based off of the inputs and outputs of the transactions used to construct the merkle tree.

When a miner receives a summarise transaction, the miner also follows the same process as they would if they received a remove transaction except that the miner also verifies the inputs and outputs of the received summarise transaction. Since the miner has to retrieve all of the transactions during the verification, the process for verifying the inputs and outputs is performed in the same manner as the user. If the calculated inputs and outputs do not match those in the received summarise transaction, the entire transaction is discarded.

## 3.5 Cleaning Period

The proposed blockchain in this paper would likely be continuously receiving transactions, some of which may be required to be removed at some point in the future. For example, multiple users may decide to remove transactions from the blockchain within a short time of each other. Consequently, it may be possible that there are two or more transactions that need to be removed from the blockchain at the same time. Given a transaction id, locating the corresponding transaction on the blockchain takes time linear to the amount of transactions on the blockchain since a miner has to traverse through, on average, half of the transactions to find the one with the matching id. With blockchains of larger sizes, it would be difficult for a miner to be able to find a given transaction and remove it before another user requests a transaction to be removed. To address the challenge posed by



the inefficiency of locating single transactions on the blockchain continuously, the concept of a *cleaning period* is introduced. The cleaning period is a predefined interval during which the miner will remove from the blockchain all transactions due for removal. Locating a single transaction still requires linear time but since non append operations on the blockchain are now batched into one single cleaning period, multiple operations can be performed in one iteration over the blockchain.

The implementation of the blockchain described in this paper spawns a new thread every cleaning period to encourage task concurrency in the miner process. Since the average time it takes to locate any transaction on the blockchain is linear to the number of transactions that have been mined, the time it takes to remove a transaction on the blockchain will increase over time, assuming that the miner is continuously receiving and mining transactions. As a result, the operations on the blockchain (such as removing temporary transactions) will take longer every cleaning period and so the threads spawned every cleaning period will live longer on average. With blockchains of larger sizes, this will present a problem since it will be likely that there are many threads attempting to iterate over the blockchain to search for transactions that need to be operated on. If there is a large number of threads trying to access the database storing the blockchain, the overall performance of the system may degrade because the processor has to be distributed amongst a larger number of threads and because the amount of threads may cause lock contention in the database if multiple threads try to access the database concurrently. Additionally, thrashing may occur if there is a large enough number of threads spawned by the miner process such that the memory consumed by a subset of the spawned threads will constantly cause page faults. To mitigate or prevent the potential problems caused by an increased runtime in the threads spawned every cleaning period, a dynamic interval between cleaning periods is used instead of a fixed amount with the interval steadily increasing proportionally to the size of the blockchain. This will allow for threads spawned each cleaning period to finish or iterate through more of the blockchain before another thread is spawned in the next cleaning period.

One drawback to using a dynamic cleaning interval is that it will grow indefinitely as the blockchain grows in size, meaning that at very large sizes of the blockchain, it could

potentially be several hours or even days before each cleaning period. The consequence of having a very large cleaning interval is it allows more temporary and summarisable transactions to accumulate before being purged. The benefit of removing transactions is diminished as well because of the increased time it takes to locate a transaction off the blockchain. To address this, a tradeoff between storage reclamation and processing time is made. Beyond a predefined limit of blockchain size, the cleaning interval will be capped. When the cleaning interval is capped, only a set amount of the most recent blocks will be checked to determine whether they contain the transactions corresponding to the transaction ids sent in a user's remove or summarise transaction. This way, users still have the ability to remove transactions from the blockchain but only for a certain period of time. Temporary and summarisable transactions are not affected by this cap because their block hashes of the blocks they are contained in are stored, allowing constant time removal of those types of transactions.

## **3.6 Correctness**

To test that the behaviour of all the modules used in this system is correct, a script is used to individually test all the different functionality provided by the miner. The transactions received and mined, number of blocks generated are checked against expected results. See Appendix C for the code used to test the miner's processing correctness.

## **3.7 Performance Indicators**

To determine the effectiveness of the blockchain implemented, several metrics will be used, most of which relate to the storage used in the blockchain. The storage footprint is one such indicator of performance of the blockchain; by creating a blockchain with the same number of blocks but differing numbers of various transaction types, it can be measured how much storage the blockchain consumes versus other combinations. For example, to determine how much storage reclamation there is when using temporary transactions, a certain percentage of transactions can be generated of the temporary type with the rest of them being permanent after which the temporary transactions can be removed from the blockchain during a cleaning period. The storage footprint after removing temporary transactions can be compared to the original storage footprint before transaction removal.

The same method can be applied for summary transactions or even a combination of the different transaction types.

The other major indicator of feasibility of the system outlined in this paper is runtime. There is extra computation required when processing the storage flexible transactions, including searching for transactions to delete and summarising transactions so the runtime of utilising different transaction types versus purely permanent types (as a traditional blockchain would use) can be compared.

Additionally, cost of blockchain storage could be measured. Omaar calculates the cost of storing data permanently on the blockchain [33]. An approximate figure of \$100 per GB to store data on the blockchain is given; a similar method to calculate the cost of storing data on the flexible blockchain could be used or to calculate a dollar amount saved from the storage reclaimed when transactions are pruned or summarised. Assuming a length of "permanence" to be 25 years, the storage cost of one gigabyte is \$4 per year. Alternatively the figure of \$0.04 per gigabyte can be used to calculate the cost savings using this system [8]. The cost savings can then be compared to the cost of extra processing time required due to the additional transaction types introduced in this paper. Origin Energy prices one kilowatt hour at 31 cents per hour [34]. Jaiahtilal et al estimate that a computer's power usage at 164 Watts [35], pricing one hour of computation at

$$164W \cdot \frac{\$0.31}{1000Wh} \approx \frac{\$0.05}{h}$$

The resources consumed by the miner can be monitored and used to measure the performance as well. The computational resources could be used to indicate whether the system is scalable. The two resources whose usage will be monitored over the lifetime of the miner will be the amount of memory used and the amount of threads running at any time. If too much memory is consumed at any time the miner is running, it could possibly cause the miner to crash from a lack of memory. If there are too many threads running at the same time, the process performance can degrade significantly due to the overhead of context switching and also the possibility of thrashing.

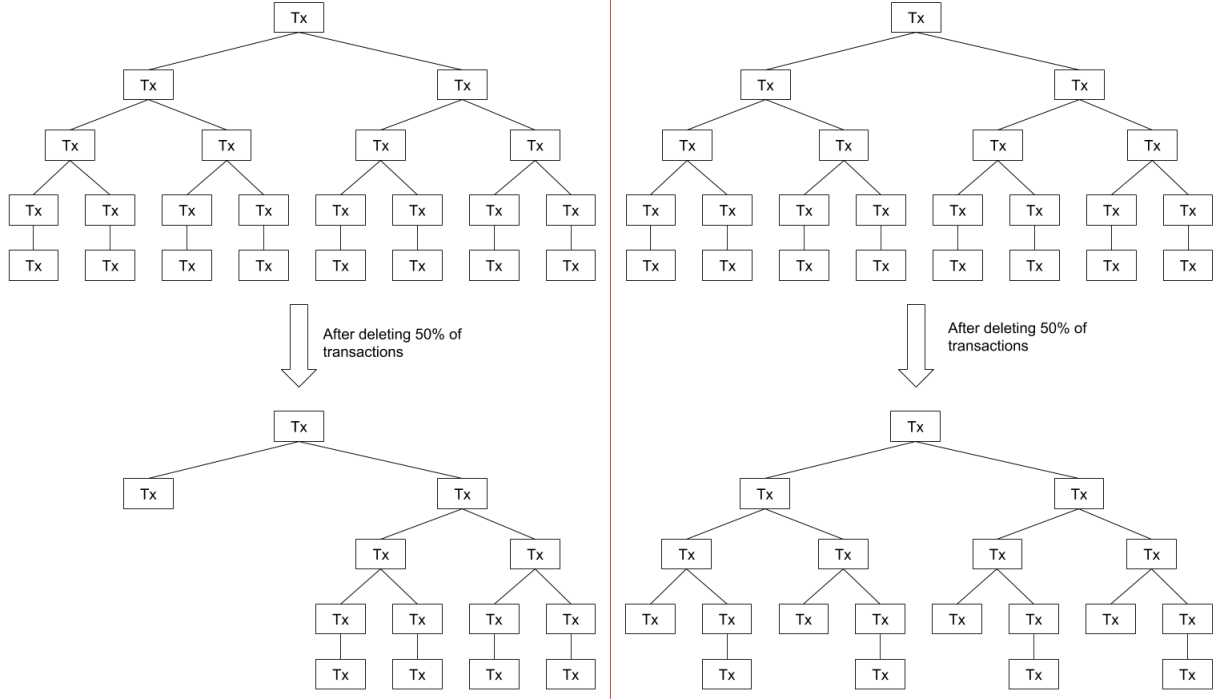
The aforementioned metrics can then be applied to different sized blockchains to determine how they scale and whether it affects the feasibility at larger scales.

## 4 Evaluation

The system described in this paper was simulated using a one miner one node configuration on a 4 core machine with 8 gigabytes of RAM. The system supports multiple senders (and actually performs better with more than one sender) but only one was used for simplicity. Different types of transactions were sent in individual simulations to test how different transaction types would impact the processing of data by the miner.

All of the extra transaction types include removing some subset of transactions from the blockchain. Another parameter that is changed when testing this system is the location of the removed transactions. Recall from Figure 2 that if a merkle tree node's children have no children, then the node can safely remove their children. As a result, the storage savings difference can be significant when deleting consecutive leaves from the merkle tree versus deleting leaves that are not consecutive. Figure 8 illustrates this: the left tree has its first 50% of transactions deleted while the right tree has every 2nd leaf deleted. As a result, nearly the entire left half of the left tree has also been deleted, leading to an increased storage reclamation.

4 sets of tests were run, with each test sending either 150000, 375000, 750000 and 1500000 transactions (approximately 100MB, 250MB, 500MB and 1GB, respectively). The miner was timed how long it took to create the appropriate number of blocks and remove, summarise or otherwise process any received transactions. The storage footprint was measured after every test and other metrics mentioned in 3.7 were monitored throughout the lifetime of the miner process.



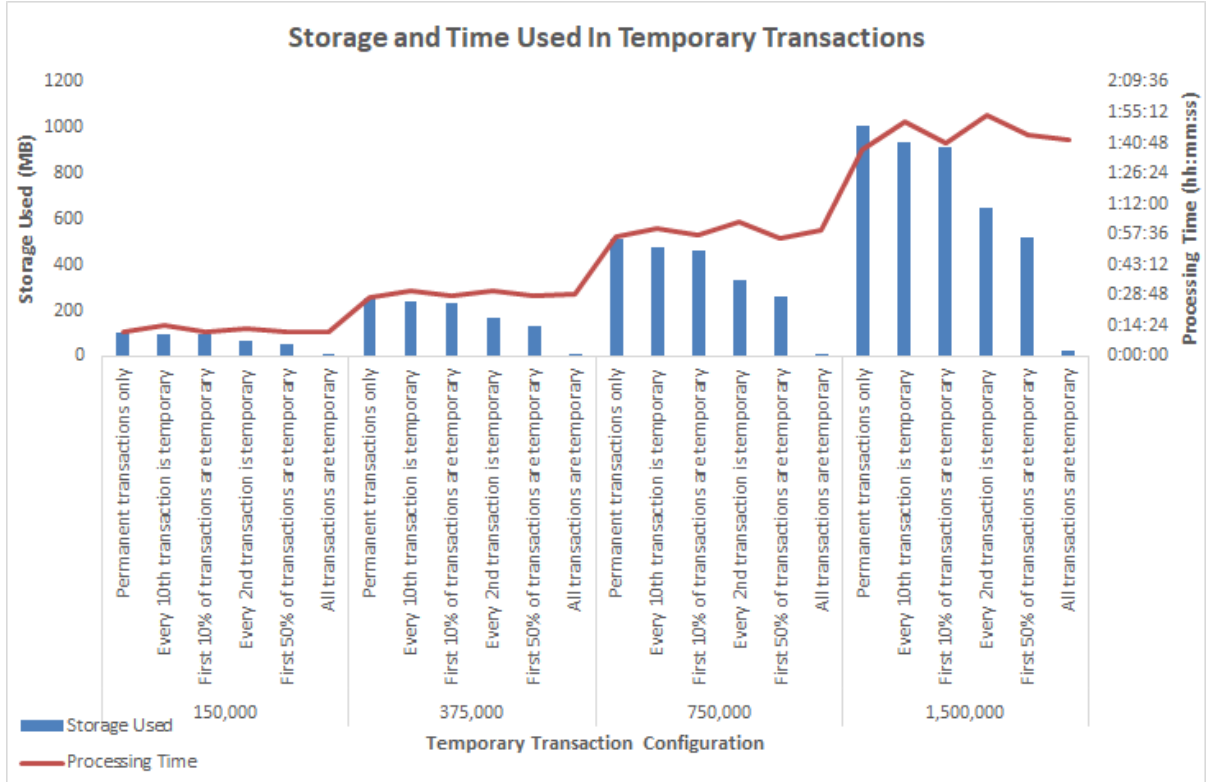
**Figure 8: The impact of transaction location when removing from the blockchain.**

## 4.1 Results

### 4.1.1 Temporary Transactions

Temporary transactions were generated with a random time to live value between 1 second and 80 seconds. Figure 9 shows the storage footprint and miner processing time of sending temporary transactions compared to a blockchain of purely permanent transactions. As expected, all configurations of temporary transactions cause some storage reduction in the blockchain. It can also be seen the impact the location of removed transactions has on the storage reduction (as mentioned above), with the difference in removing every second transaction and the first 50% of transactions becoming more noticeable as the number of transactions increases.

The processing time it takes for temporary transactions follows a general pattern, with the processing of non-consecutive temporary transactions taking longer than if the transactions were temporary. The time it takes to receive, process and finally purge temporary transactions is not significantly longer than if only permanent transactions were sent; this is especially true for temporary transactions that are consecutive in a merkle tree.



**Figure 9: The storage footprint and processing time for temporary transactions.**

The ratio of processing times of temporary transaction to permanent transactions can be used to determine the scalability of the configuration. From Figure 10, it can be observed that the time ratio remains fairly consistent even as the number of transactions scales up, remaining close to the time it takes to process purely permanent transactions. In the graph, the storage ratio is also shown, with higher amounts of temporary transactions leading to a massively reduced storage footprint in the blockchain.

The peak RAM consumption and thread count of each temporary transaction configuration was measured during each run; Figure 11 shows the results of both of these metrics. Both the peak RAM consumption and thread count slowly increase as the number of transactions sent increase. The increase can be observed over all configurations of transaction sent, implying that the increase in both of these metrics are due to an increase in processing required because of the large number of transactions received rather than a difficulty in scaling the blockchain for the case of temporary transactions.

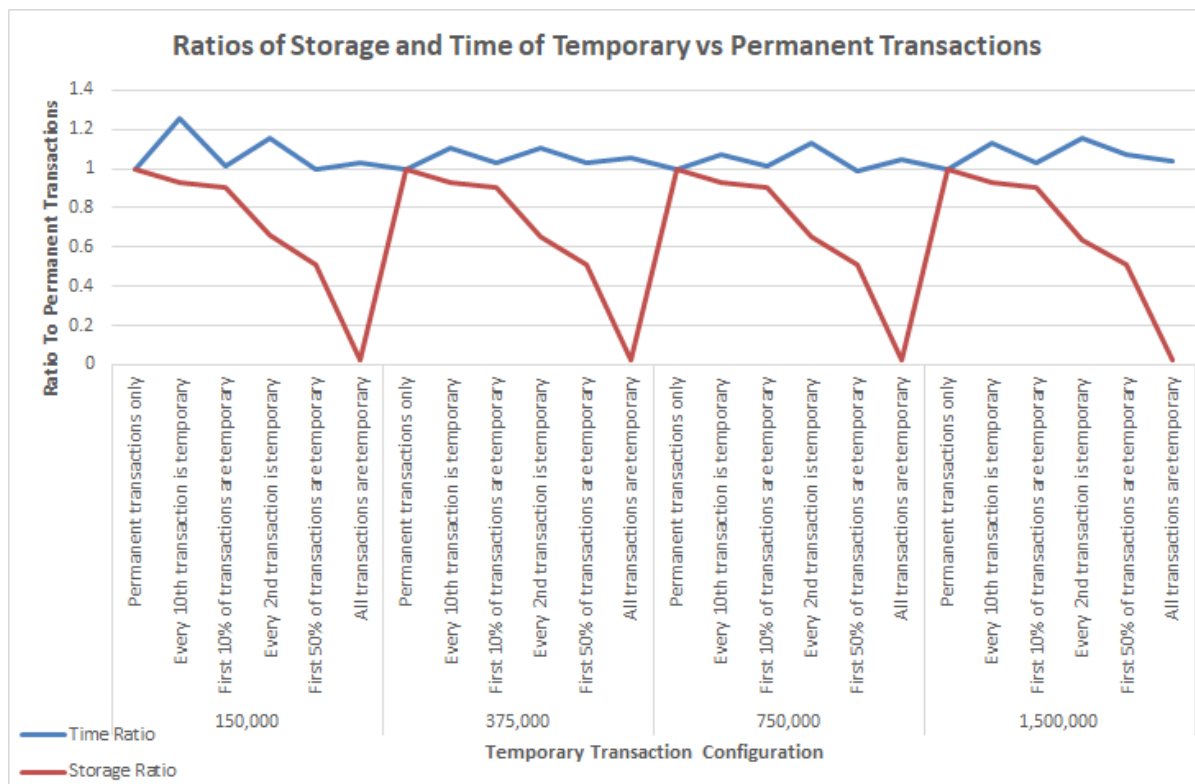


Figure 10: The ratio of processing time for temporary transactions to permanent transactions.

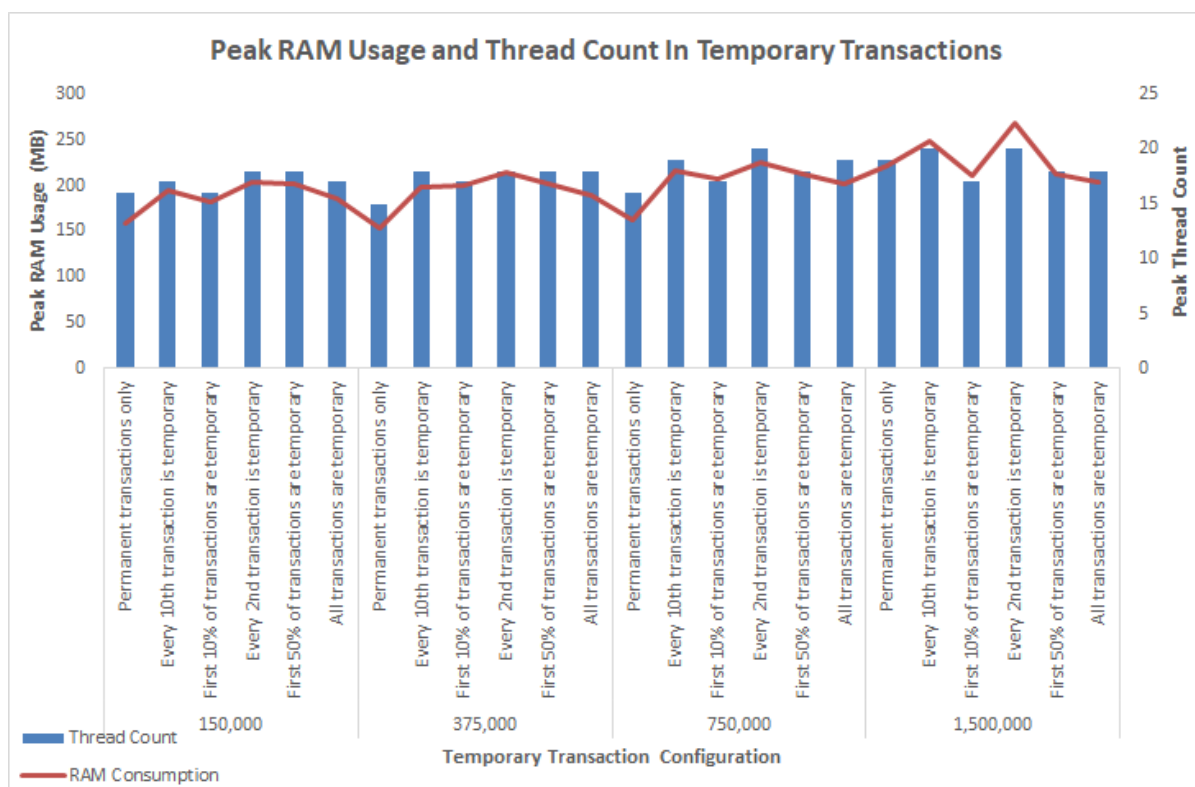
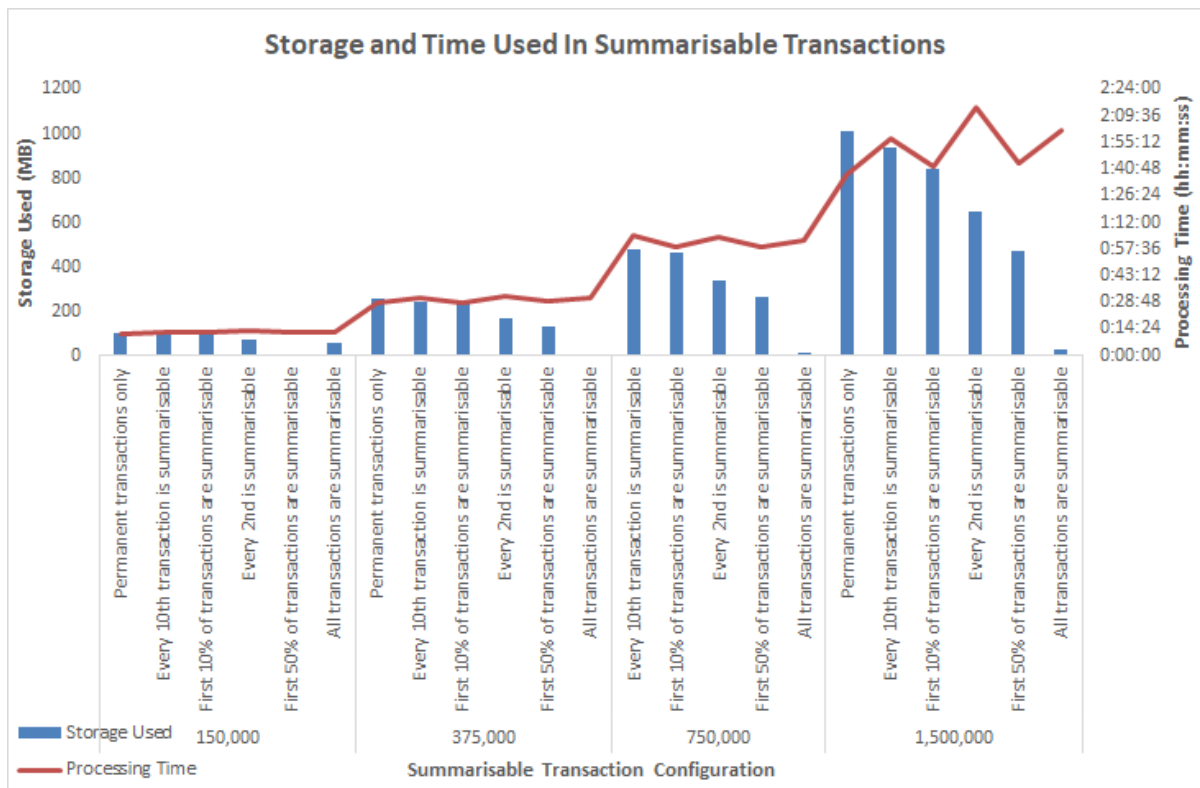


Figure 11: Peak RAM usage and thread count for temporary transactions.

### 4.1.2 Miner Summarisable Transactions

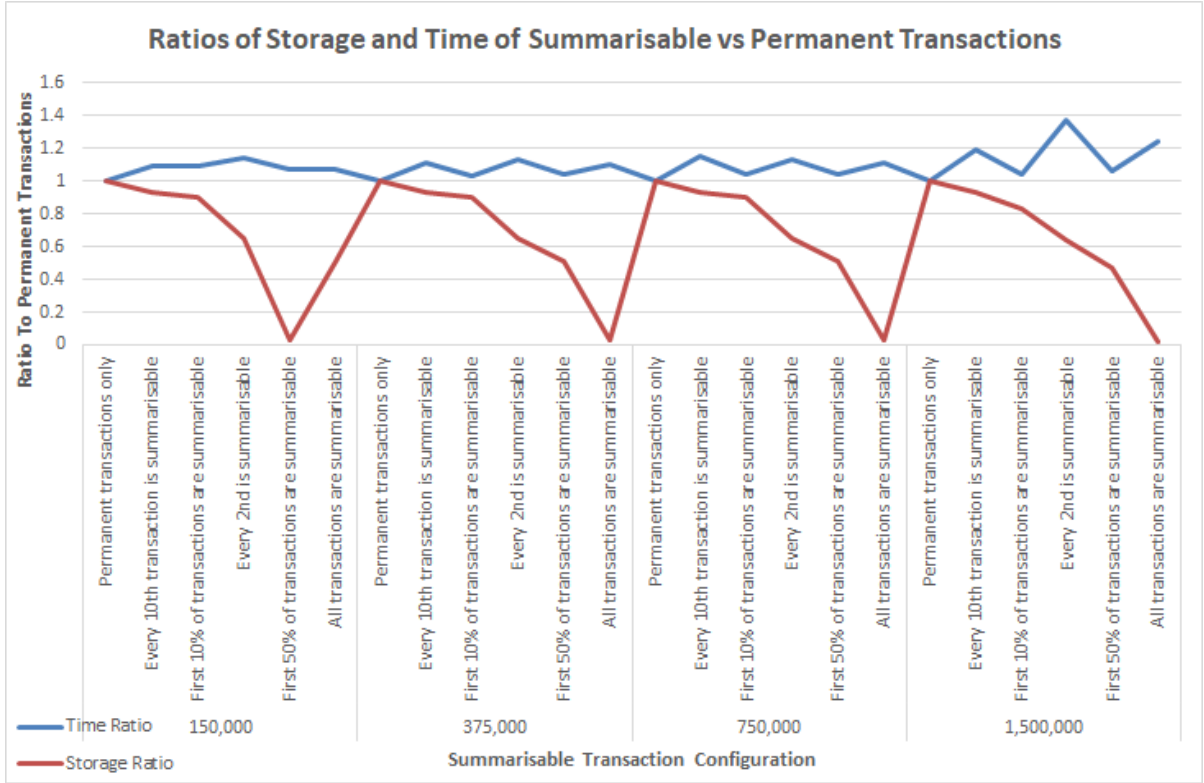
The storage used and processing time for miner summarisable transactions is shown in Figure 12. Compared to the temporary transaction graph in Figure 9, the storage footprints and processing times for summarisable transactions follow a similar pattern. Compared to temporary transactions, summarisable transactions have a slightly larger storage requirement because it requires adding additional transactions to the blockchain; the difference can be significant if there are many summarisable transactions generated however, as creating many transactions also requires creation of new blocks and their merkle trees.



**Figure 12: The storage footprint and processing time for miner summarisable transactions.**

Again, as with temporary transactions, the percentage increase in processing time of miner summarisable transactions versus permanent transactions remains fairly consistent, mostly below 20% increase in processing time. This can be seen in figure 13 alongside the storage ratio. The storage ratio also follows a pattern similar to temporary transactions, with significant storage footprint reductions the larger the amount of transactions that are summarisable and the closer the summarisable transactions are to each other in a single merkle tree.



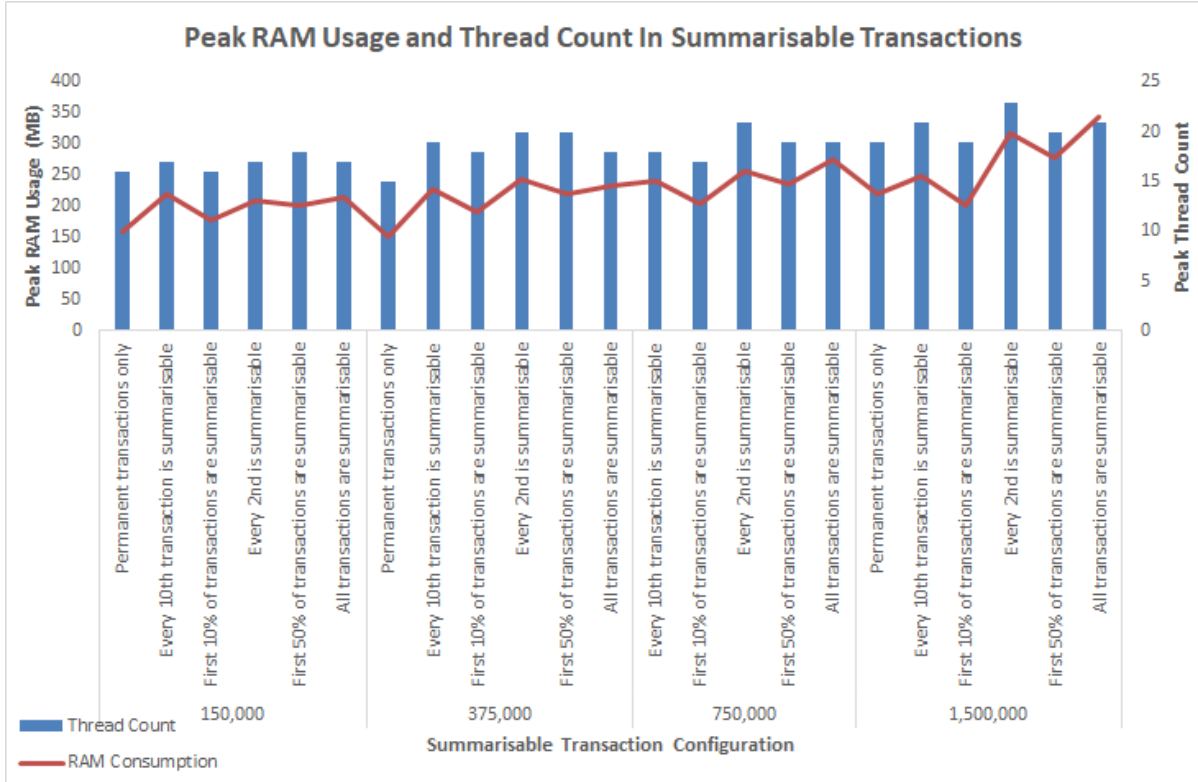


**Figure 13: The ratio of processing time for miner summarisable transactions to permanent transactions.**

The peak RAM consumption and thread count for summarisable transactions can be seen in Figure 14; there is a trend of both metrics slightly increasing linearly to the amount of transactions that the miner receives. However, comparing the cases of 150000 and 1500000 transactions sent, the difference in these metrics is not as significant as the difference in magnitude of the transactions sent, suggesting that the system is still scalable for this type of transaction.

#### 4.1.3 Remove Transactions

In Figure 15 showing the storage and processing time required for user remove transactions, the storage reclamation is less significant than temporary and miner summarisable transactions. This can be explained by the fact that for every transaction a user wishes to remove, a transaction id must be included in the remove transaction sent by the user. Additionally, there is a slight difference in both processing time and storage saved depending on how frequent the remove transaction is sent, with a decreased overall processing time and an increased storage reclamation the less frequent the remove transaction is sent.



**Figure 14: Peak RAM usage and thread count for miner summarisable transactions.**

However, this means that there are more transaction ids in a single remove transaction if the frequency is decreased because there are more accumulated before sending the transaction. This could present a challenge if a single transaction contains a significant enough amount of transaction ids that cause issues such as potential denial of service or performance degradation. This issue is not explored in this paper but a potential mitigation could be to limit the size of transactions received by the miner, limiting transaction sizes to be more reasonable.

The performance vs storage tradeoff is very noticeable in Figure 16 with remove transactions that attempt to remove all transactions sent causing a spike in the processing time with close to an 80% increased processing time for an approximate 55% saving in storage. The ratio of storage reclamation remains fairly consistent as the blockchain size increases but the processing time increases. This is likely due to the increased amount of data the miner has to iterate through when locating the appropriate transaction to remove.

As with temporary and miner summarisable transactions, the peak RAM consumption

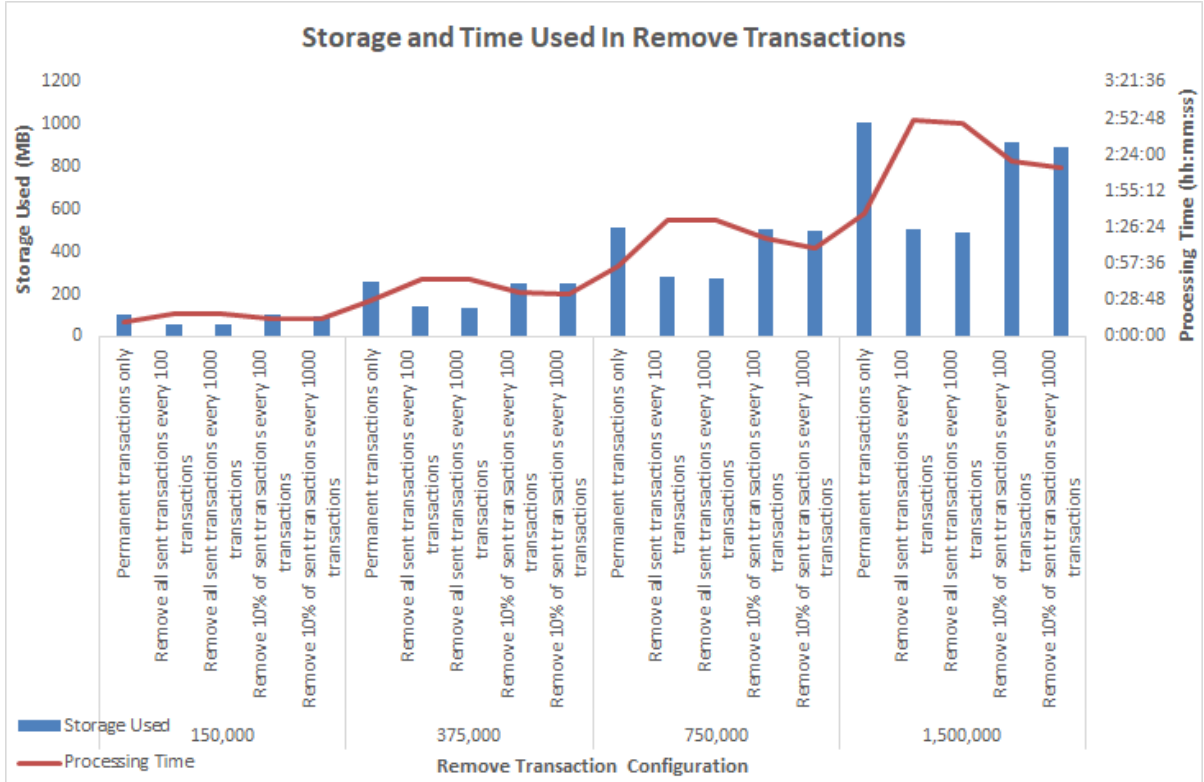
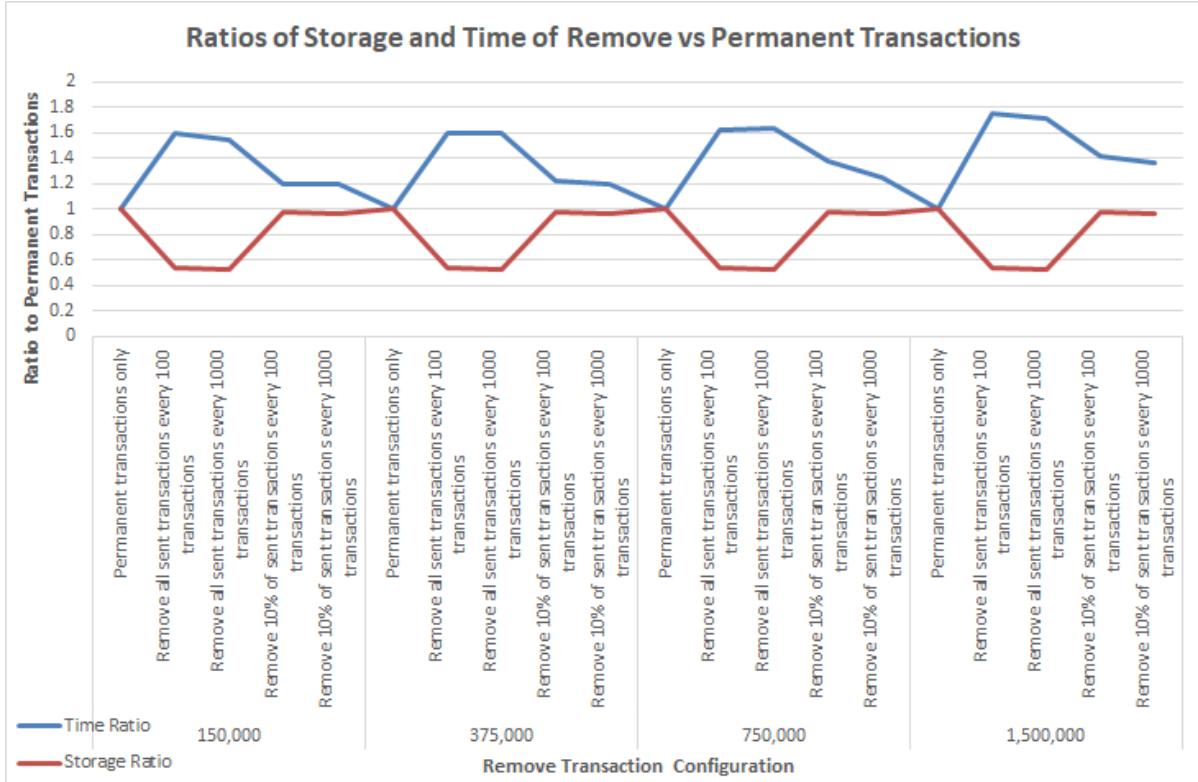


Figure 15: The storage footprint and processing time for user remove transactions.

and thread count increases with the size of the blockchain. However, the peak for RAM consumption for removing all transactions is much more significant than the previous cases; this can be explained by the fact that the transaction ids from remove transactions are stored in main memory up until the point that the transactions are (either successfully or unsuccessfully) removed from the blockchain in the next cleaning period. If the blockchain is still in the stage of having a dynamic cleaning interval, then at larger cleaning intervals more transaction ids from remove transactions can accumulate before the next cleaning period is triggered, causing an increased peak RAM consumption. The thread count is similar to temporary transactions and miner summarisable transactions however, because the number of objects stored in RAM does not directly affect the number of threads spawned.

#### 4.1.4 Summarise Transactions

The results for user summarise transactions are largely similar to remove transactions as the process is similar to remove transactions except that all processing times are slightly increased due to additional steps required during verification. All of the metrics follow



**Figure 16: The ratio of processing time for user remove transactions to permanent transactions.**

the same overall trend as remove transactions.

From Figure 18, it can be seen that the summarise transactions can overall effect some storage footprint reduction on the blockchain at the expense of an increased processing time. The storage reclamation is similar to remove transactions except with a slightly reduced storage reclamation due to the extra data stored in summarise transactions. The difference in storage reclamation is dependent on some factors however, most importantly, the size of the inputs and outputs of the transactions being summarised and the amount of unique inputs and outputs to the summarise transaction. These factors also affect miner summarisable transactions as the summarising process is the same.

The processing time and storage footprint ratios can be seen in Figure 19; compared to the remove transactions, the processing time ratio is larger and the storage footprint ratio is also larger (i.e. there is less storage reclamation) due to the factors mentioned above.

The peak RAM consumption and thread count also follow a slightly increasing trend like

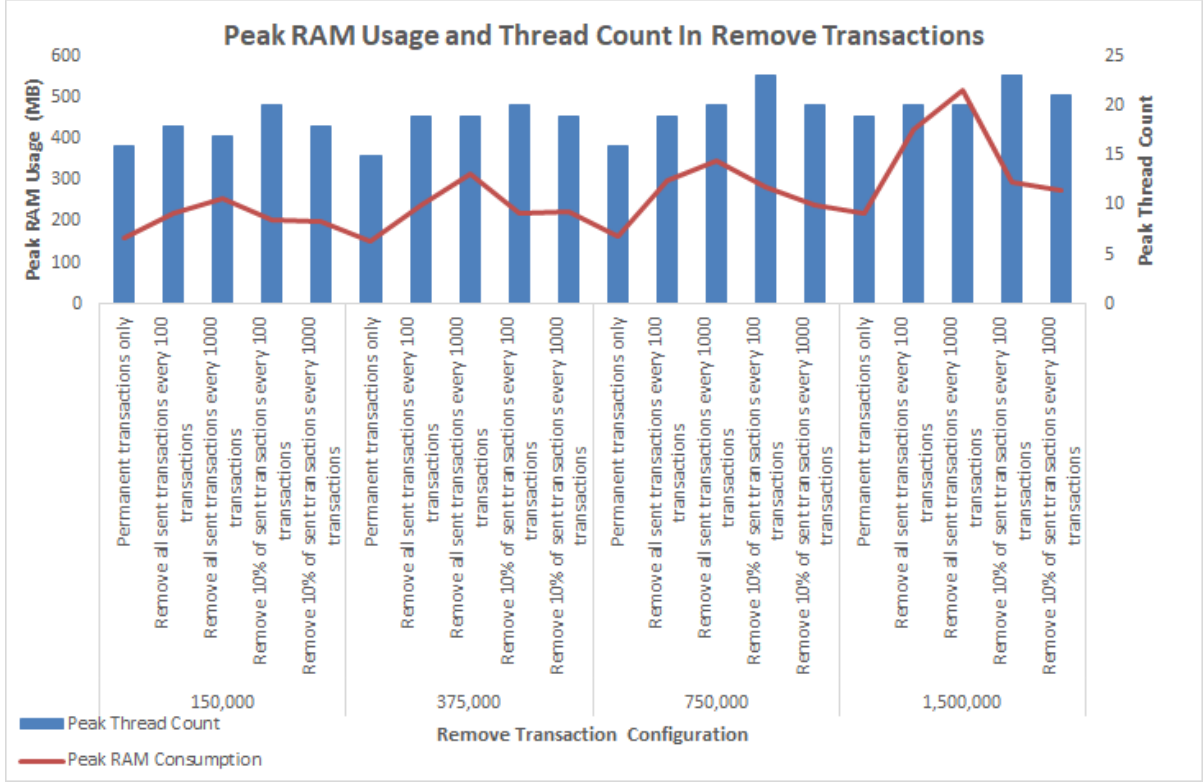


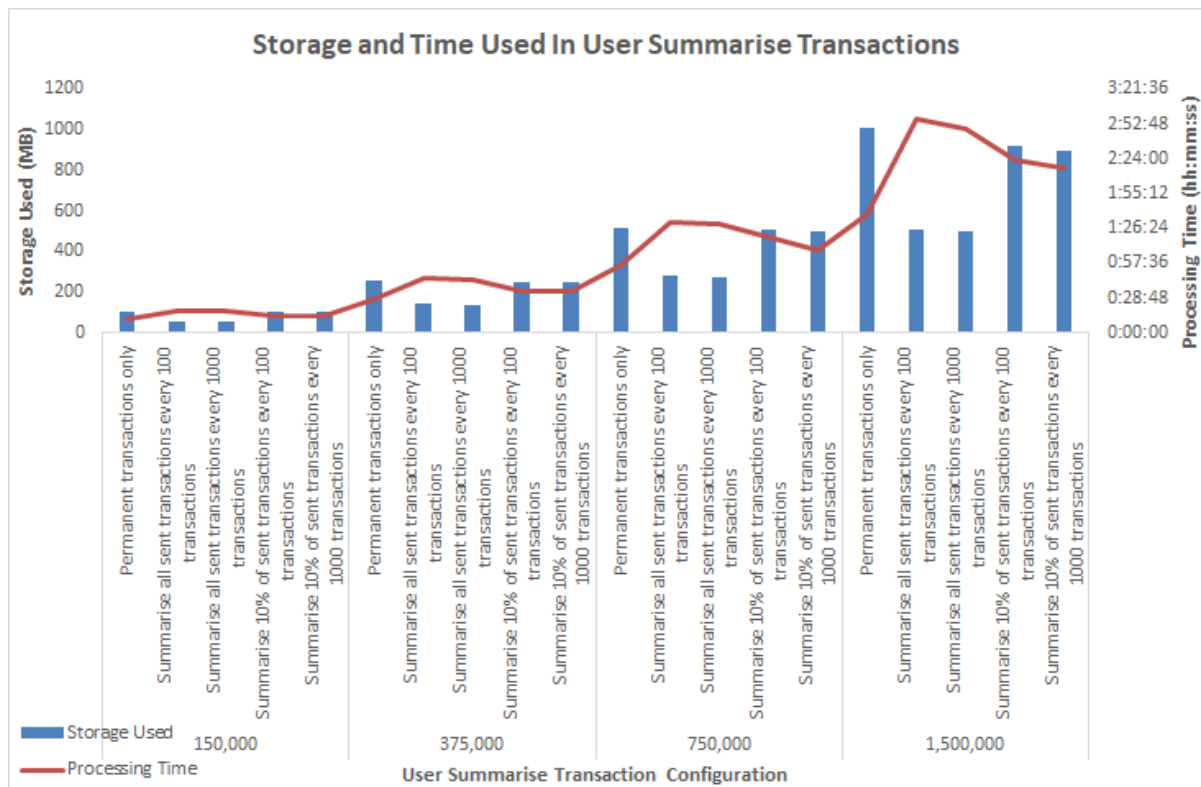
Figure 17: Peak RAM usage and thread count for user remove transactions.

the previous cases as seen in Figure 20. The largest peaks are when all the transactions sent are to be summarised, which is to be expected as was the case of remove transactions. The peak thread count is slightly higher than in remove transactions but this could be due to external factors (e.g. operating system resource management).

#### 4.1.5 Testing On Large Blockchain

The system implemented is also tested on an existing large (>100GB) blockchain to test the case where the miner has changed to a fixed cleaning period and only checks the last  $n$  blocks, where  $n$  is some constant. The effect of changing the cleaning interval and  $n$  is observed to determine how they affect the storage footprint of the blockchain.

Figure 21 shows how changing the cleaning interval affects the number of summarise transactions created by the miner. The trend is linear which can be expected if the transaction throughput remains consistent as a consistent amount of miner summarisable transactions would be received by the miner before each cleaning period starts. Since the miner summarised transactions are appended to the blockchain, a longer cleaning period corresponds to less storage consumed by the summarised transactions as there are less of



**Figure 18: The storage footprint and processing time for user summarise transactions.**

these transactions being created.

Figure 22 shows how the storage footprint of the blockchain is affected by changing the size of the cleaning interval and how many blocks are stored and checked. From the graph, it can be seen that the more blocks that are checked, the greater the storage reclamation on the blockchain but with diminishing returns. The change in cleaning interval has a larger effect on the storage reclamation; a longer cleaning interval leads to a decreased storage saving. A longer cleaning interval increases the probability that the list of blocks stored for checking during cleaning intervals is overwritten as the blockchain grows, decreasing the probability that a remove transaction is successful at removing all the transactions it sends from the blockchain. It can be seen that increasing the number of blocks stored increases the storage saving; however, increasing the blocks stored requires increasing the cleaning interval because the miner has to check a larger amount of blocks. If these values were to increase to a large amount, then the performance would essentially degrade to using a dynamic period and checking the entire blockchain which is what is trying to be avoided by fixing the cleaning interval at large blockchain sizes.

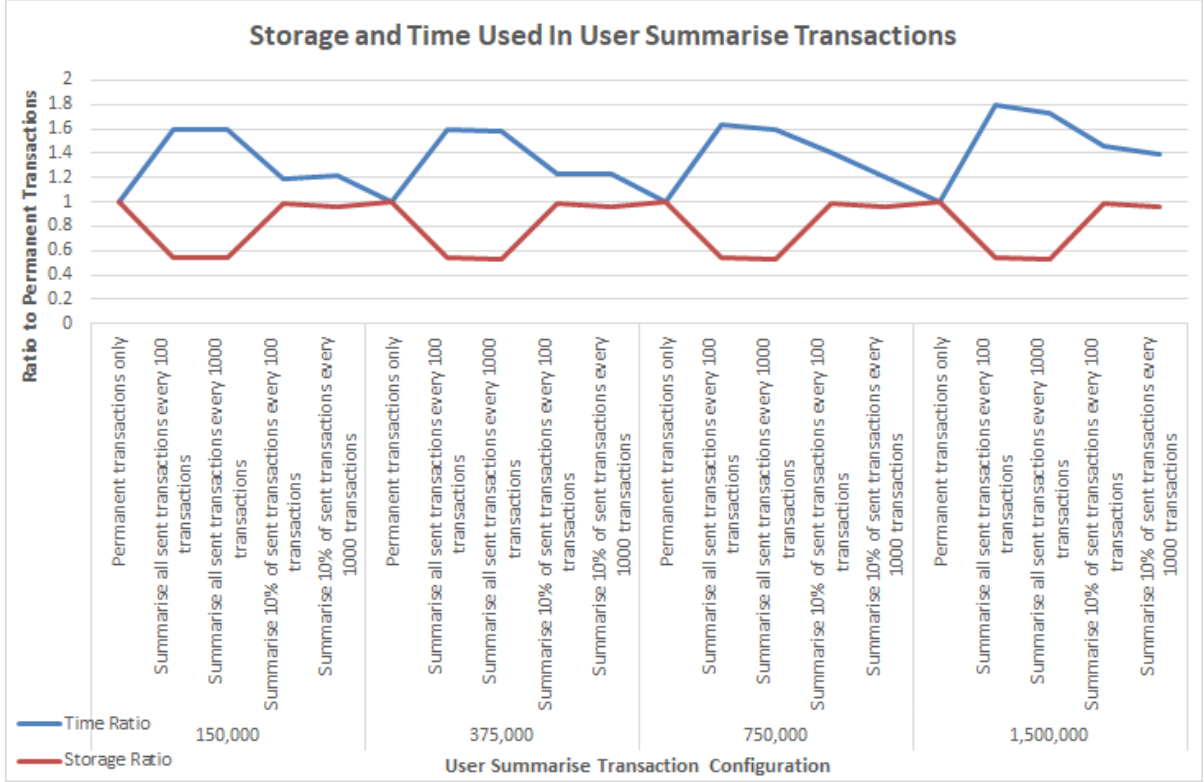


Figure 19: The ratio of processing time for user summarise transactions to permanent transactions.

## 4.2 Discussion

Using the cost of storing one gigabyte and the cost of a computer hour from 3.7, the cost benefit of this system can be estimated. To do so, the storage saving of all transaction types are scaled to 1GB and the extra processing time is scaled by the same factor.

In the case of temporary transactions, the processing time is fairly consistent, hovering at around a 10-20% increase in processing time, regardless of the size of the blockchain and the amount of transactions marked temporary. The storage reclamation however, is dependent on the amount of transactions marked temporary and can be up to a 97% decrease in the storage footprint of the blockchain if all transactions are temporary. Table 1 shows the extra processing time required to achieve a 1GB storage saving. From 3.7, one gigabyte costs around \$.04. Thus all transaction configurations for temporary transactions but having every 10th transaction as temporary have a net positive benefit in terms of cost.



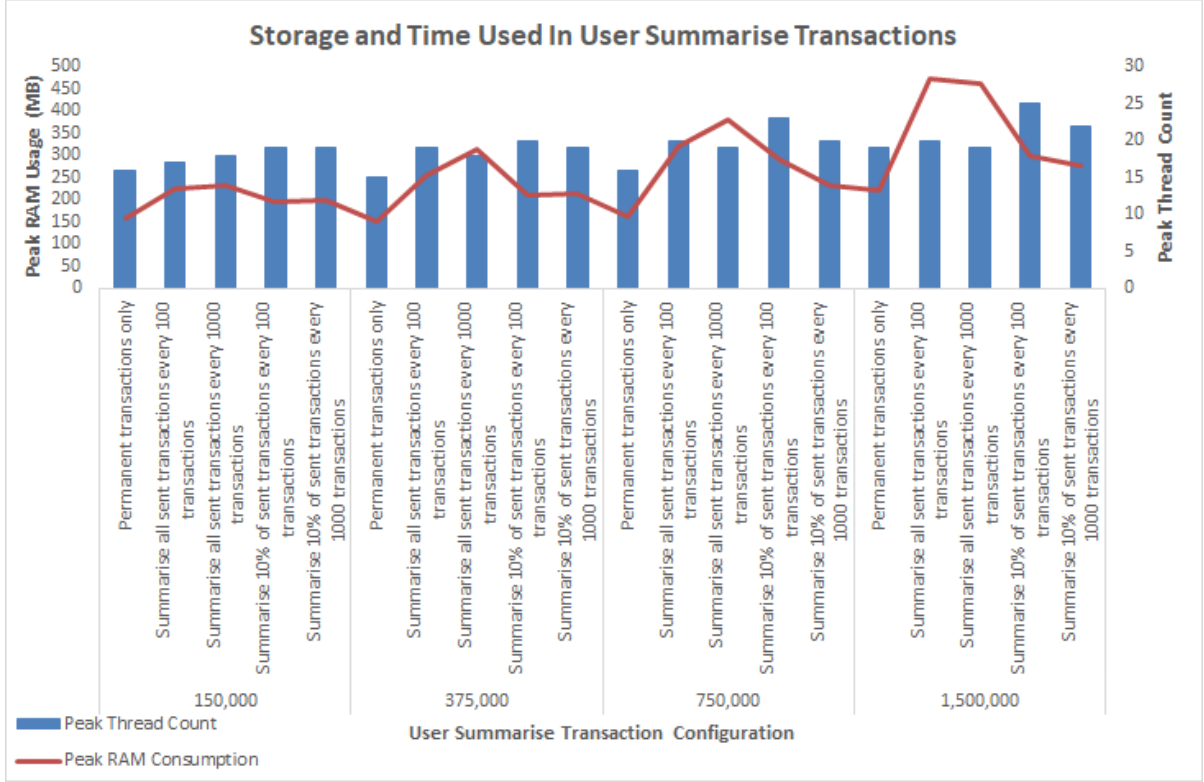


Figure 20: Peak RAM usage and thread count for user summarise transactions.

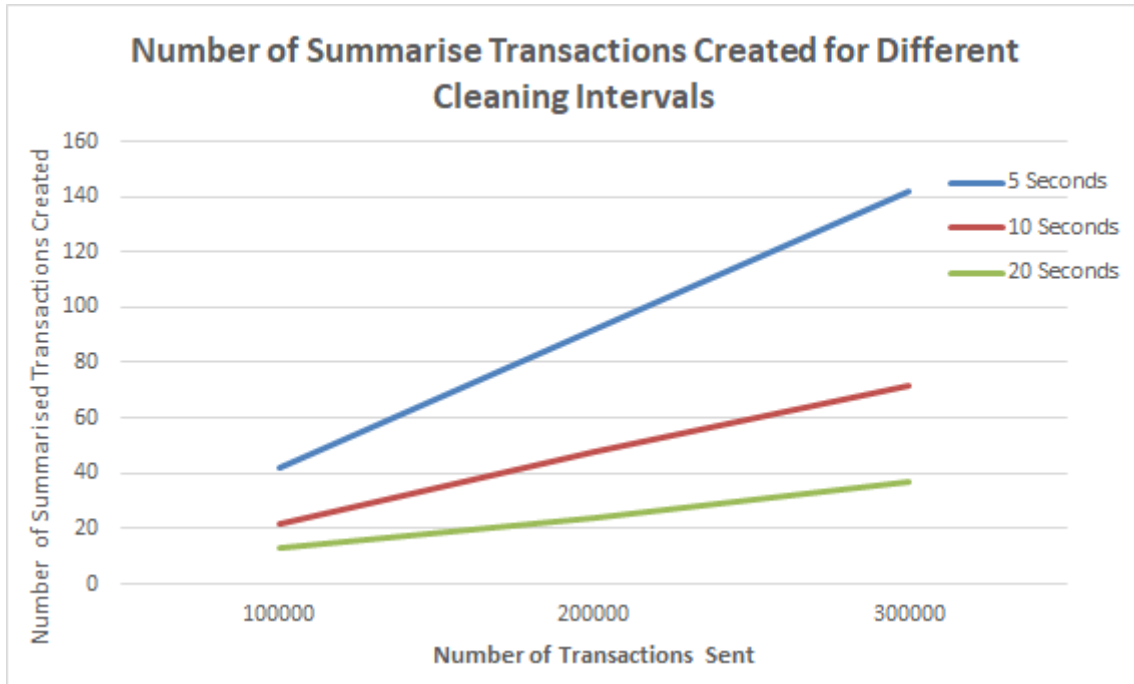
Transaction Configuration	Extra Time Required(h:mm:ss)	Cost
Every 10th transaction	2:56:22	\$0.147
First 10% of transactions	0:29:50	\$0.025
Every 2nd transaction	0:42:54	\$0.036
First 50% of transactions	0:13:48	\$0.012
All transactions	0:04:05	\$0.003

Table 1: Time and cost required to achieve a 1GB storage saving for temporary transactions

Similar to temporary transactions, miner summarisable transactions extra processing times remain fairly consistent, usually below a 20% increase over permanent transactions. From Table 2, again, all summarisable transaction configurations but having every 10th transaction as summarisable provide a net benefit in terms of cost.

The cost of remove transactions is shown in Table 3; none of the transaction configurations are profitable from a purely cost perspective. However there is an intangible benefit in that users are given the ability to delete their past transactions, providing them more privacy than what they would have in a blockchain where all transactions are permanent. Additionally, the miner can offset the cost of these remove transactions by charging the





**Figure 21: Number of summarise transactions created for different cleaning intervals.**

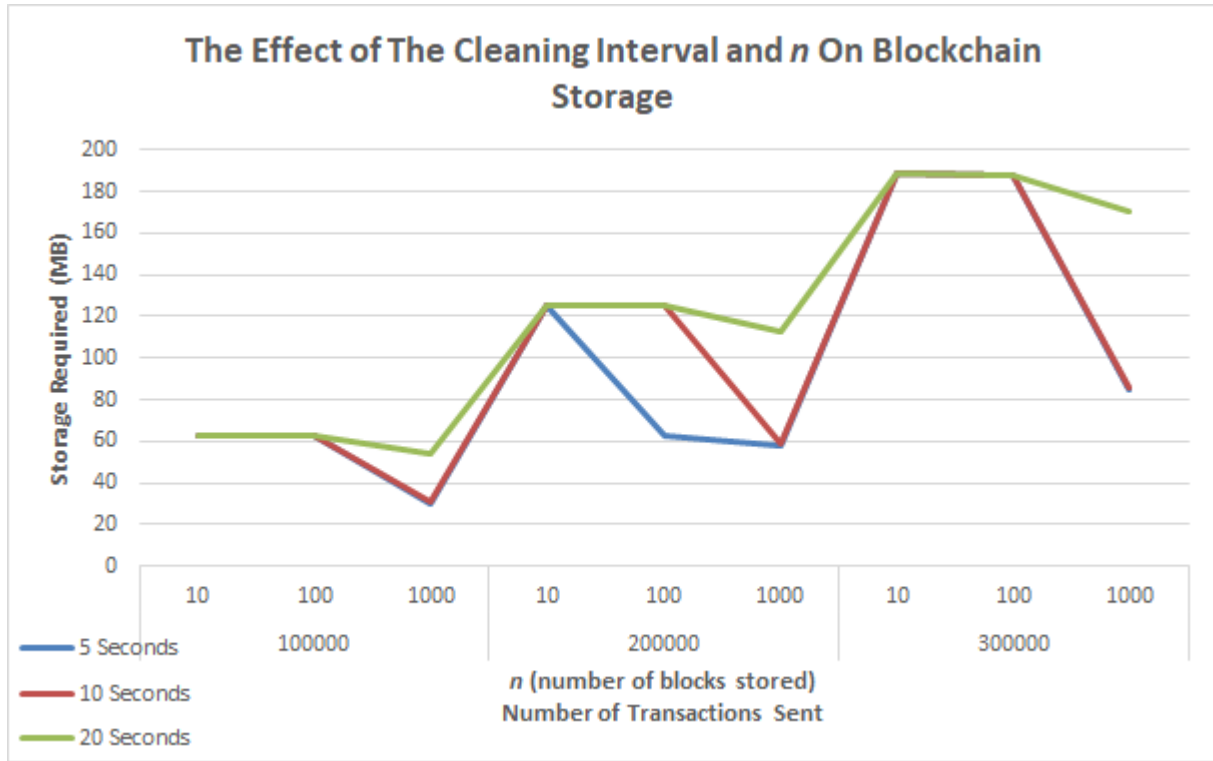
Transaction Configuration	Extra Time Required(h:mm:ss)	Cost
Every 10th transaction	4:25:52	\$0.222
First 10% of transactions	0:22:07	\$0.0184
Every 2nd transaction	1:40:07	\$0.0834
First 50% of transactions	0:11:07	\$0.009
All transactions	0:24:13	\$0.020

**Table 2: Time and cost required to achieve a 1GB storage saving for miner summarisable transactions**

user a fee for the ability to remove transactions due to the extra cost of processing.

Similar to remove transactions, user summarise transactions are not cost effective from a storage savings vs processing time perspective as can be seen in Table 4. However, the same arguments can be made that there are intangible benefits in providing users in the blockchain the flexibility in managing the storage of their past transactions and that the miners can potentially offset the costs of the extra processing time by charging some fee from the user.

However, from Chapter 3.7, the cost of storing one gigabyte for a year is \$4. The cost saving over 1 year for one gigabyte thus outweighs the cost in additional processing time



**Figure 22:** The effect of changing the cleaning interval and number of blocks on the storage requirements of the blockchain.

Transaction Configuration	Extra Time Required(h:mm:ss)	Cost
Remove all every 100 transactions	2:24:25	\$0.120
Remove all every 1000 transactions	2:16:01	\$0.113
Remove 10% every 100 transactions	7:20:47	\$0.367
Remove 10% every 1000 transactions	4:59:22	\$0.249

**Table 3:** Time and cost required to achieve a 1GB storage saving for remove transactions

to remove or summarise transactions. Additionally, the argument can be made that it is infeasible to continually add storage media as the size of the blockchain grows; there is a physical size footprint for all storage media and space is not an infinite resource. Adding more storage media also increases the energy costs of the system storing the blockchain as each individual drive requires power.

Although overall, summarising transactions (miner summarisable and user summarise) take longer to process than removable transactions (temporary and remove) and as a result, cost more to process, summarising transactions provide a benefit in that there is still some trace of the activities of users on the blockchain. When a transaction is removed, the auditability of the blockchain decreases as not every transaction can be traced back to the

Transaction Configuration	Extra Time Required(h:mm:ss)	Cost
Summarise all every 100 transactions	2:34:39	\$0.128
Summarise all every 1000 transactions	2:19:24	\$0.116
Summarise 10% every 100 transactions	8:04:52	\$0.404
Summarise 10% every 1000 transactions	5:27:14	\$0.272

**Table 4: Time and cost required to achieve a 1GB storage saving for user summarise transactions**

genesis block. Summarising transactions also decreases the auditability as transactions are still removed but there is one transaction that is added back to the blockchain that provides an overall idea of the user’s activities.

One drawback of this system is that it is partially hardware dependent. If there are any non-permanent transactions, the hard drive read/write speeds affect the overall runtime of the system. If there are user summarise or remove transactions that need to be processed, then the CPU clock speed will affect the runtime of the system as the miner has to iterate through the entire blockchain or some subset of it to search for the transactions that a user has sent in the transaction id merkle tree. Due to this, the cleaning interval is dependent on the hardware of the miner as well; if the hardware is slow, then the cleaning interval must be increased to accommodate, otherwise the performance could degrade as cleaning periods could overlap because the miner has not finished processing the data in one cleaning period before the next one has started. The system can enforce a minimum hardware requirement for miners in an attempt to mitigate this issue but then a barrier of entry could deter new users from entering the system if the requirements are too high.

Once the blockchain has reached its size limit and has switched to a fixed cleaning interval and will only check the last  $n$  blocks, the ability of removing old transactions is dramatically decreased. Depending on the value of  $n$ , users can only remove or summarise very recent transactions which may be undesirable, especially if there is a cost levied for these types of transactions. A possible solution to this could be to have a long running thread that continually iterates through the entire blockchain indefinitely and attempts to remove transactions that are not stored in the blocks that the miner will check every cleaning period.

## 5 Conclusion

In this paper, the concept of storage flexibility in a blockchain via the introduction of transaction types was presented. In a traditional blockchain, all data is stored indefinitely which could present a challenge for high volume applications. Four extra transaction types, temporary, miner summarisable, remove and user summarise, were added so that miners and users in a blockchain would have the ability to remove transactions and as a result, reclaim some of the storage used to store the removed transactions.

The method in which storage flexibility could be provided was described and it was shown that the method of provisioning user flexibility in the form of remove and user summarise transactions did not compromise the security of the blockchain through the use of the generator verifier.

From testing, it was shown that temporary and summarisable transactions provided a net benefit in terms of cost, with the cost savings due to a reduced storage footprint greater than the cost incurred due to the extra processing time of these transaction types. User summarise and remove transactions do not provide a cost saving as the cost of processing those transactions, including searching for the transactions, outweighs the potential storage savings. However, there are benefits to still using these transaction types such as increased privacy and providing users with extra flexibility so use of these transactions cannot be completely ruled out. Additionally, the cost of processing these transactions can be passed back onto the user so it could even be a net neutral function in terms of cost. The use of user summarise and remove transactions ultimately depends on the use case of the application; if the system wishes to be more attractive to users, then the provision of user flexibility in data storage could be worth the extra processing cost.

The scalability of using transaction types was also explored in this paper. It was shown that temporary and miner summarisable transactions add a constant overhead to the system's processing time over a system using permanent transactions. However, when the system is using a dynamic cleaning period, user summarise and remove transactions have an increasing overhead the larger the blockchain size. When the system rolls over to a

fixed cleaning interval then the runtime is only linear to the amount of remove or user summarise transactions whose throughput is capped by the hardware, thus the overhead can be constant as the blockchain size increases. Thus, the system at this stage can be considered scalable.

The value of the cleaning interval was also explored in relation to how it affected the blockchain storage. A dynamic cleaning interval is used for small sizes of the blockchain when it is feasible to iterate through the entire blockchain when searching for transactions to remove but at large sizes of the blockchain where it is infeasible, a fixed cleaning interval is used instead. For fixed cleaning intervals, only a subset of the blockchain can be searched and the consequences of searching only the most recent blocks is discussed. The drawbacks of a fixed cleaning interval is also discussed, with an issue being that it either restricts miners to have a minimum hardware requirement in order to meet the performance demands of the system, or to have a longer cleaning interval which has its own drawbacks.

This paper assumed that all participants in the system are honest; however, this is not necessarily the case in the real world. In the case of blockchain, a consensus protocol is used to prevent malicious nodes from affecting the blockchain. An extension to this system could potentially be to have some sort of method to prevent malicious nodes from affecting the blockchain in a negative way, whether it be from proof of work, proof of stake or some other method of reaching consensus amongst miners.

The implementation of this system was done in Python which has its own drawbacks due to it being a interpreted language and the use of the global interpreter lock preventing taking advantage of multiple cores. Future work could implement this system in a compiled language where concurrency control is explicitly managed to achieve better performance and determine again the feasibility of a storage flexible blockchain.

## 6 References

- [1] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” , 2009.
- [2] R. Merkle, “Method of providing digital signatures,” Patent US4 309 569A, 01 05, 1982. [Online]. Available: <https://patents.google.com/patent/US4309569A/en>
- [3] M. Crosby, Nachiappan, P. Pattanayak, S. Verma, and V. Kalyanaraman, “Blockchain technology: Beyond bitcoin,” *Applied Innovation Review*, no. 2, 2016.
- [4] PwC, “Malta’s new regulatory framework built for blockchain technology,” <https://www.pwc.com/mt/en/publications/technology/pwc-malta-blockchain-alert.html>.
- [5] J. Hendry, “NSW puts digital driver’s licence on a blockchain,” <https://www.itnews.com.au/news/nsw-puts-digital-drivers-licence-on-a-blockchain-512298>.
- [6] G. Nott, “NSW birth certificates, property titles and motor rego headed for the blockchain,” <https://www.cio.com.au/article/646468/nsw-birth-certificates-property-titles-motor-rego-headed-blockchain/>.
- [7] Q. Li, “Safeguard your blockchain solution with chip-level security alibaba cloud blockchain service,” 2018.
- [8] A. Klein, “Hard drive cost per gigabyte,” <https://www.backblaze.com/blog/hard-drive-cost-per-gigabyte/>, 2017.
- [9] “Bitcoin, Litecoin, Namecoin, Dogecoin, Peercoin, Ethereum stats,” <https://bitinfocharts.com/>, 2018, online; accessed 05-Apr-2018.
- [10] J. Howell, “Number of Connected IoT Devices Will Surge to 125 Billion by 2030, IHS Markit Says - IHS Technology,” <https://technology.ihs.com/596542/number-of-connected-iot-devices-will-surge-to-125-billion-by-2030-ihs-markit-says>, 2017, online; accessed 05-Apr-2018.
- [11] Cisco, “Cisco Global Cloud Index: Forecast and Methodology, 2016-2021 White Paper,” <https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.pdf>, 2018.

- [12] J. Poon and T. Dryja, “The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments,” <https://lightning.network/lightning-network-paper.pdf>, 2016.
- [13] Visa, “Small Business Retail,” <https://usa.visa.com/run-your-business/small-business-tools/retail.html>, n.d, Online; accessed 03-Apr-2018.
- [14] Visa, “Visa Inc. at a Glance,” <https://usa.visa.com/dam/VCOM/download/corporate/media/visa-fact-sheet-Jun2015.pdf>, 2015.
- [15] “Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation),” <https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679>.
- [16] M. Szydło, “Merkle Tree Traversal in Log Space and Time,” in *Advances in Cryptology - EUROCRYPT 2004*. Springer Berlin Heidelberg, 2004, pp. 541–554.
- [17] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. Gün Sirer, D. Song, and R. Wattenhofer, “On Scaling Decentralized Blockchains,” in *Financial Cryptography and Data Security*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 106–125.
- [18] R. Dennis, G. Owenson, and B. Aziz, “A Temporal Blockchain: A Formal Analysis,” in *Conference: 2016 International Conference on Collaboration Technologies and Systems (CTS)*, 2016, pp. 430–437.
- [19] R. K. Raman and L. R. Varshney, “Dynamic distributed storage for scaling blockchains,” *CoRR*, vol. abs/1711.07617, 2017. [Online]. Available: <http://arxiv.org/abs/1711.07617>
- [20] J. Bruce, “The Mini-Blockchain Scheme,” <http://cryptonite.info/files/mbc-scheme-rev3.pdf>, 2017.
- [21] G. Ateniese, B. Magri, D. Venturi, and E. Andrade, “Redactable Blockchain or Rewriting History in Bitcoin and Friends,” pp. 111–126, 04 2017.

- [22] M. Bennett, P. Matzke, J. Hoppermann, M. Glenn, and I. McPherson, “Dont Dismiss Accentures Blockchain Redaction Solution You May Need It One Day,” <https://reprints.forrester.com/#!/assets/2/77/RES137814/reports>, 2017.
- [23] V. Buterin, “CRITICAL UPDATE Re: DAO Vulnerability,” <https://blog.ethereum.org/2016/06/17/critical-update-re-dao-vulnerability/>, 2016.
- [24] V. Buterin, “Hard Fork Completed,” <https://blog.ethereum.org/2016/07/20/hard-fork-completed/>, 2016.
- [25] M. d. Castillo, “The DAO Attacked: Code Issue Leads to \$ 60 Million Ether Theft,” <https://www.coindesk.com/dao-attacked-code-issue-leads-60-million-ether-theft/>, 2016.
- [26] A. Dorri, S. Kanhere, and R. Jurdak, “MOF-BC: A Memory Optimized and Flexible BlockChain for Large Scale Networks,” 2018.
- [27] A. Nayak, A. Poriya, and D. Poojary, “Type of NOSQL Databases and its Comparison with Relational Databases,” <https://research.ijais.org/volume5/number4/ijais12-450888.pdf>, 2013.
- [28] B. Alex, “LmdbJava Benchmarks,” <https://github.com/lmdbjava/benchmarks/blob/master/results/20160710/README.md>, 2016.
- [29] Facebook, “Performance Benchmarks,” <https://github.com/facebook/rocksdb/wiki/performance-benchmarks>, Online; accessed 11-Apr-2018”, year =.
- [30] Facebook, “RocksDB FAQ,” <https://github.com/facebook/rocksdb/wiki/RocksDB-FAQ>.
- [31] Google, “leveldb,” <https://github.com/google/leveldb/blob/master/doc/index.md>.
- [32] J. Patarin and A. Montreuil, “Benes and butterfly schemes revisited,” Cryptology ePrint Archive, Report 2005/004, 2005, <https://eprint.iacr.org/2005/004>.
- [33] J. Omaar, “Forever Isnt Free: The Cost of Storage on a Blockchain Database,” <https://medium.com/ipdb-blog/forever-isnt-free-the-cost-of-storage-on-a-blockchain-database-59003f63e01>.



- [34] Origin Energy, “Nsw residential energy price fact sheet,” [https://www.originenergy.com.au/content/dam/origin/residential/docs/energy-price-fact-sheets/nsw/1July2017/NSW\\_Electricity\\_Residential\\_AusGrid-Origin%20Supply.PDF](https://www.originenergy.com.au/content/dam/origin/residential/docs/energy-price-fact-sheets/nsw/1July2017/NSW_Electricity_Residential_AusGrid-Origin%20Supply.PDF), 2018.
- [35] A. Jaianlal, Y. Jiang, and S. Mishra, “Modeling cpu energy consumption for energy efficient scheduling,” in *Proceedings of the 1st Workshop on Green Computing*, ser. GCM '10. New York, NY, USA: ACM, 2010, pp. 10–15.

# Appendices

## Appendix A Pseudocode for summarising a list of transactions

```
# given a list of transactions X
# produce a summarised transaction Y
inputs = set()
outputs = set()
for each tx in X:
    for each input in tx:
        inputs.add(input)
    for each output in tx:
        outputs.add(output)
# take the difference between sets
distinct_inputs = inputs - outputs
distinct_outputs = outputs - inputs
Y = new transaction(distinct_inputs, distinct_outputs)
```

## Appendix B Pseudocode for verifying the generator verifier

```
# Creating the gv for a transaction tx
# secret is the generator verifier secret of the user creating tx
tx_verifier = SHA256(secret + tx.tx_id)
tx.gv = encrypt(tx.tx_id, tx_verifier)

# let X be the remove or summarise transaction received by the miner
# let Y be a list of transactions corresponding to the transaction
# ids in X's merkle tree
# To verify X
verified = True
for each tx_id, index in X:
    verifier = X.verifiers[index]
    corresponding_tx = Y[index]
    if corresponding_tx.tx_id != decrypt(corresponding_tx.gv,
        verifier):
        verified = False
        break
```

## Appendix C The miner testing code

```
#!/usr/bin/python3
from subprocess import Popen
```

```

import time
import node
import plyvel
import pickle
import random
import string

# Get a random 10 character string
def get_random_str():
    return ''.join([random.choice(string.ascii_letters
                                + string.digits) for _ in range(10)])

for i in range(5):
    Popen(["./miner.py"])

    time.sleep(5)

    blocks_to_create = 1
    saved_txs = []
    txs_per_block = 10
    tx_created = blocks_to_create * txs_per_block
    blocks_created = 0

    created = []
    ids = []
    sending_node = node.Node()
    import random, string
    print('Node sending')
    last = 1
    # Test storage as well as temporary transactions
    for i in range(1, tx_created+1):
        if i % 2 == 0:
            # Temporary transactions should be removed from the
            # blockchain so shouldn't exist
            sending_node.create_and_send_tx(get_random_str(),
                                           get_random_str(), tx_type="temp", ttl=20)
        else:
            tx_id = sending_node.create_and_send_tx(
                get_random_str(), get_random_str(), 'perm')
            saved_txs.append(tx_id)
    blocks_created += blocks_to_create

    # Test miner summarised transactions
    # Use incrementing numbers so we can test if summarise
    # works properly as well
    summ_start = last
    for i in range(1, tx_created+1):

```

```

        tx_id = sending_node.create_and_send_tx(str(last),
            str(last+1), 'summ')
        last += 1
    summ_end = last
    blocks_created += blocks_to_create

# Test user summarised transactions
# The ssent transactions should not exist after summarisation
for i in range(1, tx_created+1):
    tx = sending_node.create_tx(str(last), str(last+1),
        'perm')
    created.append(tx)
    sending_node.send_tx(tx)
    last += 1
# The summarised transaction should exist on the blockchain
tx = sending_node.create_summarise_tx(created)
saved_txs.append(tx.tx_id)
sending_node.send_tx(tx)
blocks_created += blocks_to_create

# Test user remove transactions
# Removed transactions should not exist on the blockchain
to_remove_ids = []
for i in range(1, tx_created+1):
    tx_id = sending_node.create_and_send_tx(
        get_random_str(), get_random_str(), 'perm')
    to_remove_ids.append(tx_id)
tx = sending_node.create_remove_tx(to_remove_ids)
saved_txs.append(tx.tx_id)
sending_node.send_tx(tx)

# Pad the transactions so that all the sent transactions
# will be mined and exist on the blockchain
extra_transactions = 3 * blocks_to_create
if extra_transactions:
    for i in range(extra_transactions * 9):
        tx_id = sending_node.create_and_send_tx(
            get_random_str(), get_random_str(), 'perm')
        saved_txs.append(tx_id)
print('Finished sending')
sending_node.close()

# Let the miner finish running including it's cleaning periods
time.sleep(60)
passed = True
summarise_exists = False
all_txs = []
# Kill the miner process so other processes can access the

```

```

# blockchain
Popen(["pkill", "miner"])
time.sleep(3)
db = pyvel.DB("/home/ben/mof-bc")
num_blocks = 0
# Get all the transactions that currently exist on the
# blockchain
with db.iterator() as it:
    for k,v in it:
        if k == b'last':
            continue
        block = pickle.loads(v)
        all_txs.extend(block.get_block_txs())
        # Track how many blocks have been created on the
        # blockchain
        num_blocks += 1
db.close()

try:
    # Add two because there is a 'root' block and a 'last' block
    # The 'last' block is to track state
    assert(num_blocks == blocks_created +
           extra_transactions + 2)
except AssertionError:
    print("Existing number of blocks does not equal
          number of blocks created")
    passed = False
for tx in all_txs:
    if tx.tx_type == 'summarised' and
       tx.input == str(summ_start)
       and tx.output == str(summ_end):
        summarise_exists = True
# Check if a miner summarised transaction exists on the
# blockchain
# summ transactions were sent and a cleaning period was
# allowed to run
# So at least one summarised transaction should exist
try:
    assert(summarise_exists)
except AssertionError as ae:
    passed = False
    print("Miner summarised transaction does not exist
          on the blockchain")
# Check all the transactions on the blockchain match the
# transactions that were sent
try:
    assert(set([tx.tx_id for tx in all_txs if
                tx.tx_type != 'summarised']) == set(saved_txs))

```

```

except AssertionError as ae:
    bc_ids = [tx.tx_id for tx in all_txs]
    difference = set(bc_ids) ^ set(saved_txs)
    time.sleep(1)
    # Check if the difference is because of miner summarised transactions
    for tx_id in difference:
        if tx_id in bc_ids:
            index = bc_ids.index(tx_id)
            if all_txs[index].tx_type != 'summarised':
                passed = False
        else:
            passed = False
    if not passed:
        print("Difference in transactions on blockchain and
              transactions sent")
Popen([ './rm_lvldb.sh '])
if passed:
    print("All tests passed")
else:
    print("Failed")
    break

```