Enterprise Service Chain

Dr. Leszek Rychlewski
BioInfoBank Institute

Jacek Zemło
BioInfoBank Institute

Maciej Kaźmierczyk
BioInfoBank Institute

June, 2017

Abstract

Enterprise Service Chain (ESC) is a block chain based software tool facilitating high volumes of simple transactions. In most cases, these transactions involve sending tokens between accounts, as is done with other crypto currencies. The name is derived from the concept of the Enterprise Service Bus[1] in which a crypto currency is used as the protocol of communication.

In ESC, the accounts are organized in hierarchical fashion. Each account is bound to a local node and can be addressed by an unsigned integer smaller than 2^{32}. The number of nodes is limited to 2^{16}. Blocks of transactions are created in regular intervals based on the "Proof of Stake" principle, whereby a small, fixed number of VIP nodes decide on the composition of transactions included in the block. After each block is closed the set of VIP nodes is selected based on the highest sum of tokens stored in accounts of the nodes (delegated Proof of Stake[2]). VIP nodes have total authority over nodes and accounts. If a consensus between VIP nodes is reached, other connected nodes can be partially paralyzed and their accounts modified.

Each node collects transactions signed by local accounts, combines them into a message and submits the message to the block chain network. Nodes that submit conflicting messages (double spend) are penalized heavily: permission from the master VIP node is needed to restart the double spending node. Messages from two different nodes can be processed in parallel without conflicts. All types of transactions, except one, require submission via local node. The only exception is the particularly slow transaction of "recovery of funds" that is needed in the case of local node failure or dispute. Hierarchical organization of accounts enables the incorporation of "Know Your Customer" (KYC[3]) procedures during account creation. In contrast to most other crypto currencies, there is no intention to incorporate functionality that facilitates laundering[4] of assets.

All transactions and messages are consecutively numbered and can be processed in parallel within a block. Numbering reduces the size of message and transaction identifiers, and as a result also the network traffic.

[1]`https://en.wikipedia.org/wiki/Enterprise_service_bus`

[2]`http://bytemaster.github.io/bitshares/2015/01/04/Delegated-Proof-of-Stake-vs-Proof-of-Work/`

[3]`https://en.wikipedia.org/wiki/Know_your_customer`

[4]`https://en.wikipedia.org/wiki/Money_laundering`

An account can send funds to many accounts in a single transaction, requiring 6 bytes for the target account and 8 bytes for the amount.

The main objective of the software is to achieve the processing of a continuous flow of transactions in the range of 10 k to 100 k transactions per second. At this volume it is unrealistic to expect new nodes added to the network to be able to process all transactions from the first (genesis) block before actively participating in the network. Because of this, the concept of a genesis block has been dropped and nodes have the choice to start from the current database snapshot, or to download transactions from the last stored state. To speed up the synchronization from the last state, slow nodes can ignore the verification of some signatures if the messages have been already verified by VIP nodes. To facilitate instant connection, the current state of all accounts is included in the block (it is hashed into the block hash). Changes in the VIP node sets are also reflected in the block hash to facilitate the verification of the network state by the account holders (clients of nodes) without the need to calculate the sums of all account balances.

Essentially all transactions and account maintenance require fees. Less than 10% of fees are sent to accounts of node managers. The remaining revenues are distributed as dividends across all account holders. The cash flows from dividends can be used to estimate the value of the tokens managed by ESC. The tokens can therefore be viewed as shares of stocks of ESC as an enterprise. Fees are kept constant to stabilize the valuation.

The main features of the Enterprise Service Chain can be summarized as follows:

- Delegated Proof of Stake as the block consensus mechanism to reduce network maintenance costs;
- Compact account and transaction identifiers, a reduced transaction set and parallel processing of transactions to facilitate high transaction volumes;
- Nodes are heavily penalized for double spends so most transactions can be trusted almost instantly;
- A small set of VIP nodes is responsible for network integrity, to facilitate incorporation of slow nodes with reduced transaction processing capabilities;
- A hash of all accounts is part of the block enabling instant synchronization with the block chain;
- Hierarchical organization of accounts and nodes facilitates KYC, AML[5], eID[6] supply and governance;
- Dividend payments are made to account holders and node managers to support the growth of the economy of the ESC system.

1 Introduction

The motivation for the development of ESC is the business case of a distributed payment system for advertisements on web pages. In an ecosystem of thousands of advertisers and thousands of publishers displaying ads from all advertisers

[5]http://www.investopedia.com/terms/a/aml.asp

[6]https://en.wikipedia.org/wiki/Electronic_identification

there are millions of resulting individual payments. A central institution managing the cash flow can reduce the number of payments back to thousands, but the institutions active on the ads market charge large fees – often close to 50% of the payment volume for their service[7]. Alternative approaches of ad exchange markets are prone to traffic fraud, with estimates reaching 27% of traffic to be fake. The gobal financial loss in 2016 resulting from online advertisement fraud was estimated at $7.8 G[8].

Micropayments can be implemented to address this problem, with stable advertiser – publisher partnerships being created automatically after an initial testing period that requires negligible funds and thus negligible risk. This approach requires frequent processing of a large quantity of micro transactions.

The block chain concept was intended to bypass third parties in payment networks, but the currently available implementations fail to offer the transaction volumes required by the advertisement industry. The most popular and most tested block chain (Bitcoin) offers only few transactions per second and efforts to increase this number have encountered diverse problems, due to properties intrinsic to the crypto currency's design.

A review of crypto currency systems has resulted in the conclusion that available implementations focus on features that are of little relevance for the business case of high volume micro transactions, and may in fact hinder it. Such features include:

1. Extended Privacy:
   - In most crypto currencies, anybody can create a large number of accounts without additional costs.
   - Some crypto currencies introduce additional functionality that facilitates money laundering by providing built-in procedures for mixing senders and receivers of funds.
2. Extended Functionality:
   - Some crypto currencies provide the possibility of designing and introducing automated procedures (smart contracts) to manage flows of funds stored on the block chain.
3. Incentives for Miners:
   - Most crypto currencies are based on "proof of work" which offers owners of computing hardware the possibility to obtain tokens without direct exchange of fiat.

ESC was designed to provide tools for legal payment and legal transactions. The introduction of extended privacy would support business models based on fake traffic, and could potentially attract other criminal activities such as ransom payments provoked by malicious software[9]. Extended functionality provided by some block chains offers no advantages for simple payment networks and comes

[7]http://www.iab.net/media/file/PwC_IAB_Programmatic_Study.pdf

[8]http://www.cnbc.com/2017/03/15/businesses-could-lose-164-billion-to-online-advert-fraud-in-2017.html

[9]http://www.nbcnews.com/tech/security/total-paid-malware-ransom-how-exploit-spread-n759531

with substantial risks of forks due to intentional or unintentional errors[10] in "smart contracts". Application of smart contracts requires that authors[11] and all users of the contract must be at least as smart as the contract to detect and predict failures unless a "Proof of Block Chain Author" is implemented. Such functionality can have devastating effect on the speed of transaction processing and makes any optimization of the processing procedures very difficult. The choice of "proof of work" as consensus mechanism has the economical disadvantage that all coins (tokens) available in the block chain must be indirectly bought through investments in hardware or running costs. Miners add mining hardware to the system for as long as they earn revenue, and due to global competition, the profit margins are squeezed to levels where essentially all acquired coins are worth as much as the costs of acquisition. This makes the maintenance of a payment system with a lot of stored and growing value very expensive. The revenues from maintenance don't go to the nodes that keep the network alive, but to relatively few mining pools, which leads to "node bleeding"[12].

After the review of crypto currencies, two implementations were selected as a potential basis for customizations to meet the needs of high volume micro payment system. The first choice "Bitshares" is based on the "Delegated Proof of Stake" consensus mechanism. It has extended functionality including a potentially useful built-in distributed trading platform, and claims high transaction throughput capabilities of 100 k per second. After initial analysis, the candidate was dropped because: (1) by design "Bitshares" requires the processing of the whole block chain when connecting a node, (2) we had problems compiling and connecting the node to the network, probably due to built-in time latency limits that stop synchronization when overly large transactions are encountered, and (3) the source code has over 200 k lines and safely making any modifications would require a long learning period. The second choice "Cryptonite" had the required feature of having a hash of all accounts in the block chain and the option to forget old transactions. However, it was also dropped as a candidate because: (1) after compilation and launch of a new block chain, segmentation violations were observed on some wallet transactions, and (2) because "Proof of Work" rathern than "Proof of Stake" is used as consensus mechanism and rewriting this could require complete redesign of the software.

As a result, the ESC software was written from scratch with the initial decision to use as few third-party packages as possible. The current version is written in C++, uses Ed25519 software for signatures and makes use of boost libraries. The current (academic proof of concept) version of the software[13] has fewer than 15 k lines of (admittedly messy) C++ source code and compiles in seconds. The structure of the code will be improved in the coming months before any new major additions.

[10]`https://blog.ethereum.org/2016/07/20/hard-fork-completed/`

[11]`http://www.steptoeblockchainblog.com/2017/06/my-smart-contract-just-ate-14-million-now-what-re-thinking-indemnification-for-smart-contract-risks/#page=1`

[12]`https://bravenewcoin.com/news/the-decline-in-bitcoins-full-nodes/`

[13]`https://github.com/adshares/esc`

2 Implementation

The block chain network consists of registered nodes. Each node has an id, and themaximum number of nodes is 2^{16}. Each node manages accounts - a maximum of 2^{32-1} accounts. New node registration requires a transaction from an account managed by a registered node. Each node has a weight, corresponding to the sum of balances on accounts managed by the nodes. The 32 nodes[14] with the highest weights represent the VIP nodes. VIP nodes select the transactions included in a block based on a Delegated Proof of Stake consensus mechanism. The VIP node with the highest weight is assigned the master VIP node status. The master VIP node has the power to change the secret key of nodes that committed a double spend (network crime). Nodes use their secret keys to authenticate themselves when connecting to other nodes and to sign all messages submitted to the network. Therefore a node with an invalid (changed) key is unable to connect to the network.

The block chain consists of blocks calculated at predefined intervals of 1024 seconds[15]. When a new block period starts, VIP nodes exchange information about the set of messages included in the last block. This information is expressed as the hash of all message hashes. Only VIP nodes participate in the selection procedure. The submission of the proposed hash by a VIP node is delayed depending on the weight of the node (the sum of account balances managed by the node). The node can send its proposal if it can not accept any proposal from other higher-weight nodes read from the network (usually because of missing messages), otherwise the node signs a proposal read from the network. A node can only sign one hash (vote), otherwise it will commit a double spend. The weight of the vote is equal to the weight of the node. The nodes collect votes (signed hashes) until there is a hash with enough vote weight (sum of weights of votes) so that no other hash can have higher weight or until a timeout is reached (256 seconds[16]). After a winning hash has been selected, all VIP nodes calculate the new block and send their signature of the new block to the network. Also in this case, only one signature from each node is legal. In most cases all VIP nodes should sign an identical block. In case of disagreements, nodes with different block signatures will disconnect from each other resulting in a fork. To connect again to the network a node must initiate a synchronization procedure starting from an older block with a node on the target block chain. This process currently requires manual intervention.

The Block includes:

1. current time (`uint32_t`) (*now*)
2. number of messages in block (`uint32_t`) (*msg*)
3. number of nodes (`uint32_t` even though there can only be 2^{16} nodes so `uint16_t` would suffice) (*nod*)
4. dividend for the current period (`uint32_t`) (period spans many blocks) (*div*)
5. hash of accounts (`char[32]`) (*nodhash*)
6. hash of public keys of VIP nodes (`char[32]`) (first key is from the master VIP node, other keys are sorted by node id) (*viphash*)

[14]compile parameter
[15]compile parameter
[16]compile parameter

7. hash from the previous block (`char[32]`) (*oldhash*)
8. hash of all transaction messages in this block (`char[32]`) (*msghash*)
9. final new hash of the block (`char[32]`) (*newhash*)

The new hash is calculated as follows:

```
newhash = hash2(msghash, hash2(oldhash, hash2(viphash,
        hash2(nodhash, sha256(now,msg,nod,div))
)));
where hash2(a,b) =
        sha256(a,b) if a < b,
        sha256(b,a) otherwise.
```

The addition of the viphash as a separate branch in the hash tree of the block enables account holders to monitor the changes in the composition of VIP nodes without monitoring the weights of all nodes. Assuming that changes in the weights of heaviest nodes are slow, the changes in the composition of the VIP node sets will be small, and each block will not exclude more than a few old VIP nodes from this set. Since the new set in a new block is signed by a majority of old VIP nodes, the changes in the VIP set can be validated simply by collecting the block signatures from VIP nodes without the need to process all transactions (with a matching hash-tree rooted at *msghash*), and without the need to download the states of all nodes (with a matching hash-tree rooted at *nodhash*). In the current implementation, a node client (account holder) downloads missing blocks, signatures of the latest block and signatures of all blocks where changes in VIP sets occurred.

Nodehash is the root of hashes of all nodes. The hash of a node is calculated as sha256 checksum of following elements of the node:

1. public key (used to sign messages) (`char[32]`)
2. XOR of hashes of all accounts (see below) (`char[32]`) (*hash*)
3. last message hash (`char[32]`) (*msha*)
4. id of last message (`uint32_t`) (*msid*)
5. time last message sent (`uint32_t`)
6. weight (sum of balances of accounts) (`uint64_t`)
7. status (32 bits defining the state of the node) (`uint32_t`)
8. number of accounts (`uint32_t`).

The node enumerates messages sent to the network with consecutive numbers starting from 1. The message id is incremented only for messages with regular transactions (not for messages with votes or block signatures). Messages that represent votes are treated separately. The id of the last message is stored in *msid*. The number of messages a node can send in its lifetime is currently limited to 2^{32}. Assuming a message frequency lower than 1 Hz, the size of this variable will not represent a problem for any node in this century. The new message is signed with the node's private key, and the message includes the hash of the previous message. This way each node generates its private block chain, with the block hashes stored as *msha*. The block chain can be used for forensic analysis or for auditing purpose, without the need to download all messages submitted to the network by all nodes during the investigated period. Instead of a hash tree of all accounts, an exclusive OR (XOR) is calculated using hashes of all accounts. The (cryptographic) hash of an account is constructed in a way that can not be inverted (collision resistant). The XOR of such hashes is sufficient to validate the integrity of the data. The advantage of the XOR approach is that an update of a single account requires only an update of the hash of the account, and only two XOR operations are required to update the account hash of the node. If a hash-tree would be used the update would require up to 32 hash operations (depending on the number of accounts managed by the node) and would create a more severe computational speed bottleneck than the signature verification. The disadvantage of the XOR approach is that it is impossible to generate a short hash path (branch) that is rooted in the block for the purpose of proving the state of the account. In the current implementation, a dedicated user transaction is needed to obtain the proof. This is accomplished by storing the account state in a transaction and creating a hash path to this transaction rooted at *msghash*.

Each account managed by a node has a fixed data size of 128 bytes. The fixed size of the account enables efficient storage of account data in flat binary files. The data includes following variables:

1. id of last transaction (`uint32_t`) *msid*
2. time of last transaction (`uint32_t`) *mtim*
3. public key (`char[32]`)
4. hash of last transaction (`char[32]`) *hash*
5. block time of last outgoing transaction (`uint32_t`) *lpath*
6. cousin account id (`uint32_t`) *user*
7. cousin node id (`uint16_t`) *node*
8. status of the account (`uint16_t`)
9. block time of last incoming transaction from different node (`uint32_t`) *rpath*
10. balance of the account (`uint64_t`)
11. hash of the account (`char[32]`) *csum*

The transaction id (*msid*) of the account acts as the message id of the node and is calculated as the number of previously submitted transactions plus one (*msid* starts with 1). The time of last transaction represents the time as provided by the account holder. The time must be larger than the time of the previous transaction, but it does not need to correspond to global clock time. The "hash of last transaction" field is computed using current transaction data and the hash of the previous transaction as input. As in the case of the message hash of the node (*msha*), the hash of the account represents a private block chain and can be used for auditing purpose. Using this hash, the account holder can prove that the provided set of transactions is a complete set of all outgoing transaction associated with the account. The account data is used as input for the calculation of the account hash *csum*, which employs the sha256 hashing function. The account data must be collision resistant (any two accounts must not have the same data) otherwise the XOR of account hashes would not provide proof of correctness of all accounts. To achieve this, the first message hash of the account incorporates a reference to the node id and the account id. The account data includes additional variables (*lpath*, *rpath*, *node*, *user*) representing information about changes to the account as seen by the network. These variables are used internally to manage dividend payments and account maintenance fees, along with the special transaction of recovering funds from an account in case of node failure or dispute.

The account holder can send the following transactions to the network:

1. *send_one*: send funds to one account including an optional 32 byte memo
2. *send_many*: send funds to many accounts. This is cheaper than executing many *send_one* transactions, but does not offer the option to provide a memo.
3. *broadcast*: broadcast a binary string of data that will be loaded by all nodes and provided to clients upon request. This is a method of advertising on the network.
4. *create_account*: request a local or remote node to create an additional account with the same public key
5. *create_node*: request the network to create a node and create a management account with the same public key as the issuing account
6. *retrieve_funds*: request retrieval of all funds from a remote account on a remote node with the same public key. This special transaction is deliberately very slow (in the order of weeks) and can be used if the remote node fails or refuses to send transactions from the remote account.
7. *change_account_key*: changes the public key of the account
8. *change_node_key*: changes the node key used to sign messages and connect to the network. This transaction can be issued only from the management account (account id = 0).
9. *set_account_status*: changes the status of a local account, subject to approval by the node (i.e. the node can refuse to forward this transaction)
10. *set_node_status*: changes the status of a node. Some bits of the status field require approval by the network (through voting) and some are restricted only to local changes. This transaction can be issued only from the management account (account id = 0).
11. *log_account*: request a transaction with a copy of the account data for the purpose of creating a proof of state

The set of transactions is fixed and intentionally small. This design choice enables more efficient optimization of transaction processing. Expanding the set of transactions in the future would require the update of all nodes that process all transactions (a hard fork). Additional transactions are defined for the communication between the node and the account holder (client). These transactions enable retrieving the account state (balance), the account history, the broadcast files, transaction hash paths and blocks with signatures and VIP node keys. These transactions are not part of the network protocol and can be extended at will.

All network transactions require a fixed fee plus a fee proportional to the size of the moved funds. Moving funds to accounts on remote nodes is two times as expensive as local transactions (within one node).

3 Optimization considerations

Many design choices have been made with the goal of achieving a high transaction throughput. The target of 10 k transactions per second (10 kHz) was defined as the minimum requirement. The potential to achieve a transaction frequency of 100 kHz has been set as desired target. The minimum target for the number of account has been set to 1 M with the desired target of 100 M (and the potential of 1 G accounts).

At a frequency of 10 kHz and with 100 M accounts several bottlenecks are expected:

1. CPU based limits. A single CPU core can perform around 15 k-30 k Ed25519 signature verifications per second. Because the number of transactions collected since the hypothetical genesis block would be huge, it is unrealistic to require nodes to perform a complete block-chain scan when synchronizing. This results in the requirement to store a hash of all accounts in at least some blocks. After one block (1024 seconds) the network has completed over 1 M transactions so there is the potential that all accounts have been updated. Therefore delaying the update of the account hash until only after multiple blocks have been completed would reduce the amount of CPU work needed for hashing. However, hashing the set of 100 M accounts would require several minutes and would mean that during the block a large fraction of the block time is spent on calculating the block hash. Because the hash input data during the hash tree calculation is small, replacing sha256 with a higher throughput hashing function (such as BLAKE) would not make a significant difference. To ensure a smooth transition flow between all blocks the decision was made to update the account hash after every transaction. Hashing both accounts of a transaction is expected to increase the CPU requirements of processing a transaction only by 10%, with the majority of this computational cost attributed to signature verification.

2. Disk based limits. Regular hard drives (HDDs) will handle only a few hundreds of updates per second. The only option to achieve a transaction frequency of 10 kHz using regular HDDs is to keep the databases in RAM. To achieve this, the memory occupied by the account data must be small. The choice of 128 bytes per account means that a RAM volume of around 12 Gb is needed to avoid having to keep updates of 100 M account on disks, which seems a feasible requirement for all nodes willing to access the network. Increasing the number of accounts to 1 G would require larger RAM or use of SSD disks that can handle 100 k IOPS. Such disks are currently available at a cost of below 1 k Euro.

3. Internet based limits. A regular transaction sending funds from one account to another requires at least 64 bytes to store the signature. If the target account is represented as a hash and the transferred amount requires 8 bytes, then the size of a transaction takes over 100 bytes to store. At 100 kHz a bandwidth of 10 Mbyte/s or almost 100 Mbits/s is needed just to transport the transaction data. This estimation does not include the overhead needed for the communication between peers to inform about

available transaction inventory. A continuous use of 100 Mbits/s bandwidth makes setting up a node at home problematic.

To address all 3 bottlenecks, the following design choices have been made:

1. The use of Ed25519 signatures instead of secp256k1 signatures enables the application of a batch verification procedure of transactions contained in a message, which promises a speed increase of 100% and reduces the CPU bottleneck.

2. Transactions are grouped in messages submitted by nodes. Messages from different nodes can be processed independently, which provides the benefit of exploiting multi-core or multi-processor architectures. This reduces the CPU bottleneck. Parallel processing of messages also reduces the effect of disk latencies and thus the disk bottleneck.

3. Transactions between accounts on different nodes have higher fees. This should lead to clustering of activities of a certain type within dedicated nodes, and also increase the chance of updating accounts stored on the same file page. This reduces the disk bottleneck.

4. Accounts have a small fixed size (128 B) and are well aligned with the standard 4 kB file page size. This eliminates the need of reading indexes from the disk before accessing the account data and limits the number of disk seeks.

5. Using XOR of account hashes instead of a hash tree of account hashes makes the effect of hashing of accounts negligible on the CPU performance, and eliminates the need of additional disk seeks to find the neighboring hashes for the hash tree update.

6. Using enumeration of accounts, messages and transactions enables the storage of a message id, a transaction id and the account id (address) each in a 64 bit digit (`uint64_t`). Grouping of transactions in messages and the small message id size reduces the bandwidth needed for the update of inventory changes between nodes. It also reduces the size of the message needed to describe the set of messages included in a proposed block.

7. Introduction of account creation and maintenance fees will promote the reuse of accounts and limit the number of accounts used by a client, which will reduce the required disk space and the disk bottleneck.

8. Introduction of the one-to-many transaction reduces the number signature verifications needed for a single movement of funds. Because the target account and amount can be expressed in 14 bytes (amount: 8 bytes; target node: 2 bytes; target account: 4 bytes) the bandwidth needed to inform the network about the movement is also reduced compared to standard crypto-currencies that report the target account as a 25 byte-long hash.

9. The designation of a small set of VIP nodes for the signoff of blocks provides many options to further reduce the processing requirements of other nodes in the network. The signatures of transactions are validated by the VIP nodes so there is no strict need for all nodes to verify them. This

greatly reduces the CPU requirements but does not affect the disk and network bandwidth limits. To reduce the disk bottlenecks, weak nodes can decide to ignore the information about other nodes completely and just rely on the VIP nodes to select fully validated messages. From these messages, a weak node would only need to extract the incoming transactions affecting local users. A weak node would then remain a functional node for its clients and transmit any transaction, but it would not be able to inform its clients about the states of account on other nodes. This operation mode would reduce the disk bottleneck, but would not affect the network bottleneck. A reduction of the network bottleneck is possible if a peer performs a screening of messages sent to the weak node, and filters out messages that do not affect the accounts of the weak node. The weak node can still validate the integrity of its data based on the `nodhash` (root hash of node states) stored in each block. This radical approach would make it impossible for the clients of the node to request any remote transaction transmitted over the network. Currently, only the CPU bottleneck is reduced, and nodes that lag behind with signature verification ignore it for messages validated in past blocks.

4 Results

A cluster of computing nodes with 24 cores of Intel Xeon E5-2650 v3 2.3 GHz and 196 GB of RAM was used for the tests. The initial test of cryptographic functions showed that one core of the cluster nodes is capable of performing 16 200 signature checks in single check mode and 35 300 checks in batch verification mode.

The nodes had a file server mounted synchronously over NFS which degraded the performance dramatically. To eliminate the problem of disk IO latencies, all files where kept on a local RAM disk. The network consisted of 8 nodes, with 8 threads dedicated to message processing and 16 threads for processing office requests (communication with clients sending wire transfers). The use of 8 threads for signature verifications poses a CPU-based limit for transaction processing of $8 * 35\,300\,\text{Hz} = 282\,400\,\text{Hz}$ (in batch verification mode). Four types of wire transfers were tested with 1, 10, 100 and 1000 recipients respectively. The senders and recipients where chosen randomly from the account databases. In these tests, the sender sends a batch of 1000 transactions. The tests were conducted on two database sizes: a small one with $8 * 10\,000$ accounts and larger one with $8 * 100\,000$ accounts. The performance was measured with modified network parameters, with a shortened block length of 32 seconds to reduce the time needed to accumulate results. The results are presented in the table below:

db size	**recipients**	**transactions/s**	**updates/s**
80 000	1	144 308	144 308
80 000	10	99 803	998 030
80 000	100	16 084	1 608 360
80 000	1000	1549	1 549 250
800 000	1	119 672	119 672
800 000	10	66 702	667 022
800 000	100	10 231	1 023 050
800 000	1000	1189	1 188 667

Initial results indicate that the nodes can reach a transaction processing speed of over 100 kHz. The results show that on a small database size, account update speeds of over 1 MHz can be reached, but the speed drops with increasing database sizes. This is due to the fact that transfers between different nodes are kept in RAM and are committed at block end. At high transfer speeds, most database accounts are already updated after 1 second. After the end of a block, the nodes commit all inter-node transfers but the number of such commits is not larger than the number of accounts. Local node wires are committed immediately, and due to the network design, 1/8 of all wires are local, so the minimum expected update speed at large database sizes is also above 100 kHz (1.5 MHz/8).

5 Discussion

VisaNet handles an average of 150 million transactions every day and is capable of handling more than 24 kHz (transactions per second) based on a test conducted by IBM in 2010[17]. Reports from 2015 indicate a maximum throughput of 56 kHz[18]. In 2012 Mastercard reported in its annual report[19] a capacity of 160 M transactions per hour which translates to 44 kHz. The reported throughput for the ESC platform is therefore competitive with global payment networks, whilst having the advantage of keeping a distributed ledger. The hierarchical structure of the ESC network offers the option to add weak nodes to the network that do not verify all signatures, or even that ignore many transactions.

The hierarchical structure of ESC helps to implement networks compatible with the cross-border eID concept promoted for example by the European Commission[20]. Most crypto-currencies implement schemes that do not restrict the creation of new accounts. In ESC each account is linked to a node that has the capability of modifying the status bits of the account. This helps to implement authorities or protocols that can provide reliable information about revocation of a state of an account. In any crypto-currency it is easy for the account holder to prove that a status has been assigned to the account in the past (for example by pointing to the transaction assigning the status). However, it is difficult to prove that the status has not been revoked. Testing for this is a similar problem to the detection of double-spend and requires either a trusted third party or the

[17]https://usa.visa.com/run-your-business/small-business-tools/retail.html

[18]https://usa.visa.com/dam/VCOM/download/corporate/media/visa-fact-sheet-Jun2015.pdf

[19]http://s2.q4cdn.com/242125233/files/doc_financials/annual/MA-2012-Annual-Report.PDF

[20]https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/eID

scanning of all transactions. In ESC, all (full-)nodes are capable of providing reliable information about the status of all accounts, and can act as suitable third parties. As an example, a node could offer eIDs for businesses operating in the jurisdiction of the node (e.g. a node representing a micro-nation) and assign a bankruptcy status to the account of the business if the business filed for bankruptcy protection.

ESC was designed to provide an infrastructure for fast transactions and extended governance. ESC was not designed to provide extended functionality, for example in the forms of automated procedures ("smart contracts"). It is difficult to implement such procedures on the network level without sacrificing speed but it is not difficult to implement them on the node level. Nodes can easily implement two factor authentication or multi signature wallets. Nodes can also implement exchanges of the main token with other tokens. Nodes could be responsible to guarantee the authenticity of listed tokens (for example Fiat) that they offer in trades. This could be used by micro nations to offer shares of companies registered in their jurisdiction, or digital currencies minted by the nations. Nodes could also offer additional services such as loans or insurances to local users, and in this way promote local economic growth. To improve the security of local procedures and transactions, a node could be audited by other nodes that are respected by the network. In the case of serious disputes, a misbehaving node could be excluded from the network by the VIP nodes. This would not necessarily mean a complete loss of the ability to continue business for local users, but would require the implementation of other means to exchange locally and globally accepted assets.

Recently a similar project (EOS.IO[21]) based on delegate proof of stake has been announced. The EOS.IO project aims at combining parallel processing with user defined procedures (smart contracts). There are similarities between EOS.IO and ESC, but the main difference is that ESC tries to separate the automated procedures, advanced user management (permissions, mandatory delays in transactions, recovery of stolen keys, etc.) and complicated governance models from the basic functionality that is mandatory to facilitate a high speed block-chain. The separation will enable independent development of enhanced functionality without the need of hard forks. The design choices made in ESC focus on speed and interoperability with other block chains. The interoperability objective is for example one of the reasons to keep much larger block (over 15 minutes vs. 3 seconds in EOS.IO). The detailed comparison between the two block chains would require lengthier discussion with the additional difficulty that EOS.IO is still a concept with no published implementation.

6 Acknowledgements

The project was funded by the grant PICO-2015-PL.

[21] `https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md`