NEM

Technical Reference

Version 1.2.1

February 23, 2018

Contents

Preface

> You miss 100% of the shots you don't take.
>
> - *Wayne Gretzky*

NEM is a movement that aims to empower individuals by creating a new economy based on the principles of decentralization, financial freedom, and equality of opportunity.

We would like to thank the contributors and the many people who have inspired us...

BloodyRookie gimre Jaguar0625 Makoto

1 Introduction

> He'd say hello and introduce himself, but most of the cats turned a deaf ear, pretending they couldn't hear him, or stare right through him.
>
> - *Haruki Murakami*

NEM, in its most basic form, is a crypto currency that is built on block chain technology. The NEM block chain is an improvement on existing block chain technologies. It integrates concepts from other cryptocurrencies (e.g. Bitcoin) and academic research in network theory.

NEM's primary contribution to the crypto currency landscape is a new consensus mechanism called Proof of Importance (PoI). Unlike Proof of Work (PoW), it is environmentally sustainable and does not require large scale computing resources in perpetuity. PoI is similar to Proof of Stake (PoS) except that it is not solely derived from the size of an account's balance. It incorporates other behaviors that are believed to be positive for the holistic economy. In this way, it attempts to reward active economy participants at the expense of inactive ones and dampens the rich getting richer effect that is inherent to PoS.

NEM's vision is to be the foundation of a vibrant crypto currency ecosystem that emphasizes security and trustless computing. NEM was launched with built-in support for multisig transactions and encrypted messages. Additionally, the peer-to-peer (P2P) NEM network implements a modified version of Eigentrust++ to identify and minimize the impact of malicious nodes.

NEM is evolving and this is just the beginning. Stay tuned for more things to come.

2 Accounts and Addresses

> No wind serves him who addresses his voyage to no certain port.
>
> - *Michel de Montaigne*

NEM uses elliptic curve cryptography to ensure confidentiality, authenticity and non-repudiability of all transactions. Each account is a private+public Ed25519 keypair (section 3: Cryptography) and is associated with a mutable state that is updated when transactions are accepted by the network. Accounts are identified by NEM addresses, which are derived in part from one way mutations of Ed25519 public keys.

2.1 Account state

The state associated with each account includes the following items:

- account balance
- number of harvested blocks (see subsection 5.3: Block creation)
- height of the first transaction that referenced the account
- list of multisig accounts and list of cosignatories (see subsection 4.3: Multisig related transaction types)
- information about delegated account status (see subsection 4.2: Importance transfer transactions)
- importance and NCD aware rank (see section 7: Proof-of-Importance)
- vested balance (crucial for PoI and NEM itself)

The underlying crypto currency of the NEM network is called XEM. Each account's XEM balance is split into two parts: vested and unvested.

Whenever an account receives XEM, the new XEM are added to the account's unvested balance. When an account sends XEM, XEMs are taken from both the vested and the unvested balance, to retain the vested to unvested ratio[1]. Additionally, every 1440 blocks, $\frac{1}{10}$ of the unvested balance is moved to the vested part.

[1] Of course that is not always possible

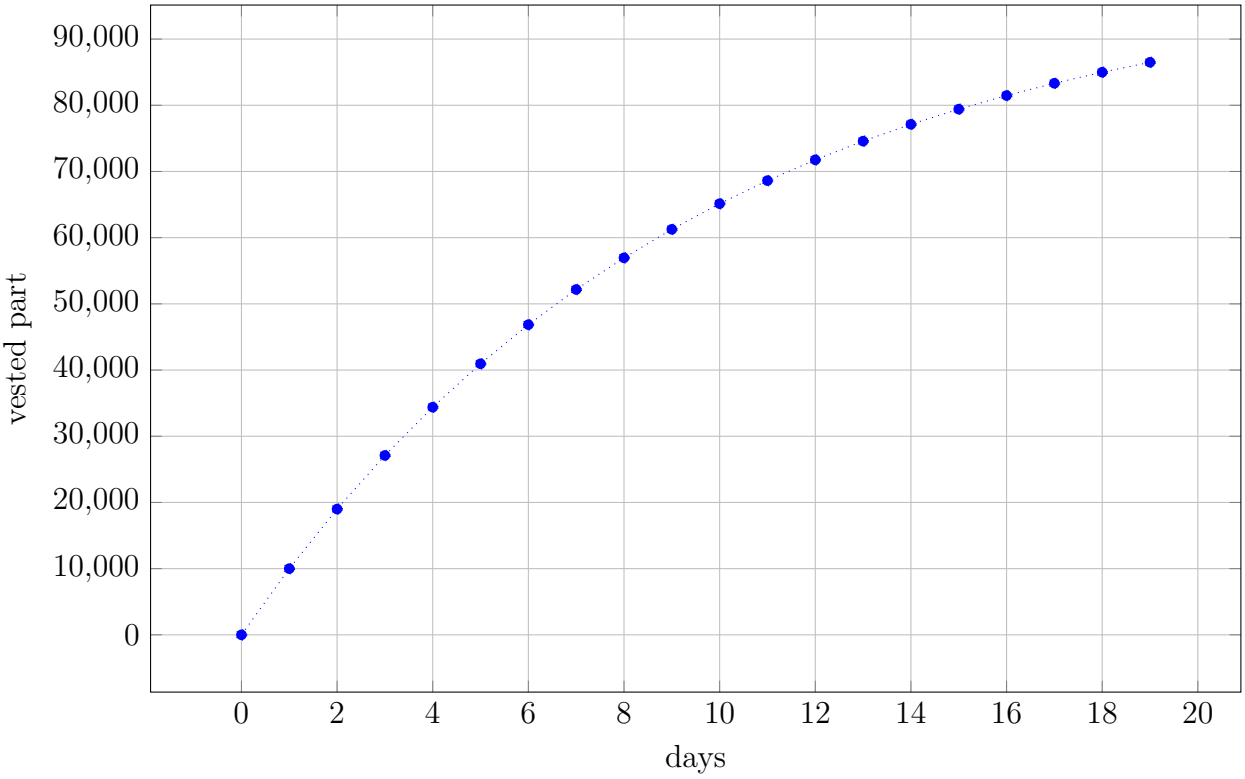


Figure 1: *Vesting of 100,000 XEM*

All accounts in the nemesis block[2] are fully vested.

2.2 NEM addresses

A *NEM address* is a base-32[3] encoded triplet consisting of:

- network byte
- 160-bit hash of the account's public key
- 4 byte checksum

The checksum allows for quick recognition of mistyped addresses. It is possible to send XEM to any valid address even if the address has not previously participated in any

[2] first block in the NEM block chain
[3] http://en.wikipedia.org/wiki/Base32

transaction. If nobody owns the private key of the account to which the XEM is sent, the XEM is most likely lost forever.

2.3 Converting a public key into an address

In order to convert a public key to an address, the following steps are performed:

1. Perform 256-bit Sha3 on the public key
2. Perform 160-bit Ripemd of hash resulting from step 1.
3. Prepend version byte to Ripemd hash (either 0x68 or 0x98)
4. Perform 256-bit Sha3 on the result, take the first four bytes as a checksum
5. Concatenate output of step 3 and the checksum from step 4
6. Encode result using base32

Example:

1. public key
   X: `deb73ed7d0334e983701feba4599a37fb62e862e45368525b8d9fb9ab80aa57e`
   Y: `169318abc3e5b002059a396d4cf1c3d35ba022c675b15fb1c4943f7662eef268`
   Z: `a90573bd221a3ae33fec5d4efc4fa137897a40347eeafe87bee5d67ae5b4f725`
2. compressed public key:
   `c5247738c3a510fb6c11413331d8a47764f6e78ffcdb02b6878d5dd3b77f38ed`
3. sha3-256:
   `70c9dcf696b2ad92dbb9b52ceb33ec0eda5bfdb7052df4914c0919caddb9dfcf`
4. ripemd: `1f142c5ea4853063ed6dc3c13aaa8257cd7daf11`
5. prepend version: `681f142c5ea4853063ed6dc3c13aaa8257cd7daf11`
6. sha3-256 of above:
   `09132a5ea90ab7fa077847a699b4199691b4130f66876254eadd70ae459dbb53`
7. 4-byte checksum: `09132a5e` (first 4 bytes of the above)
8. binary address: `681f142c5ea4853063ed6dc3c13aaa8257cd7daf1109132a5e`
9. base-32 encoding: `NAPRILC6USCTAY7NNXB4COVKQJL427NPCEERGKS6`
10. pretty-print: NAPRIL-C6USCT-AY7NNX-B4COVK-QJL427-NPCEER-GKS6

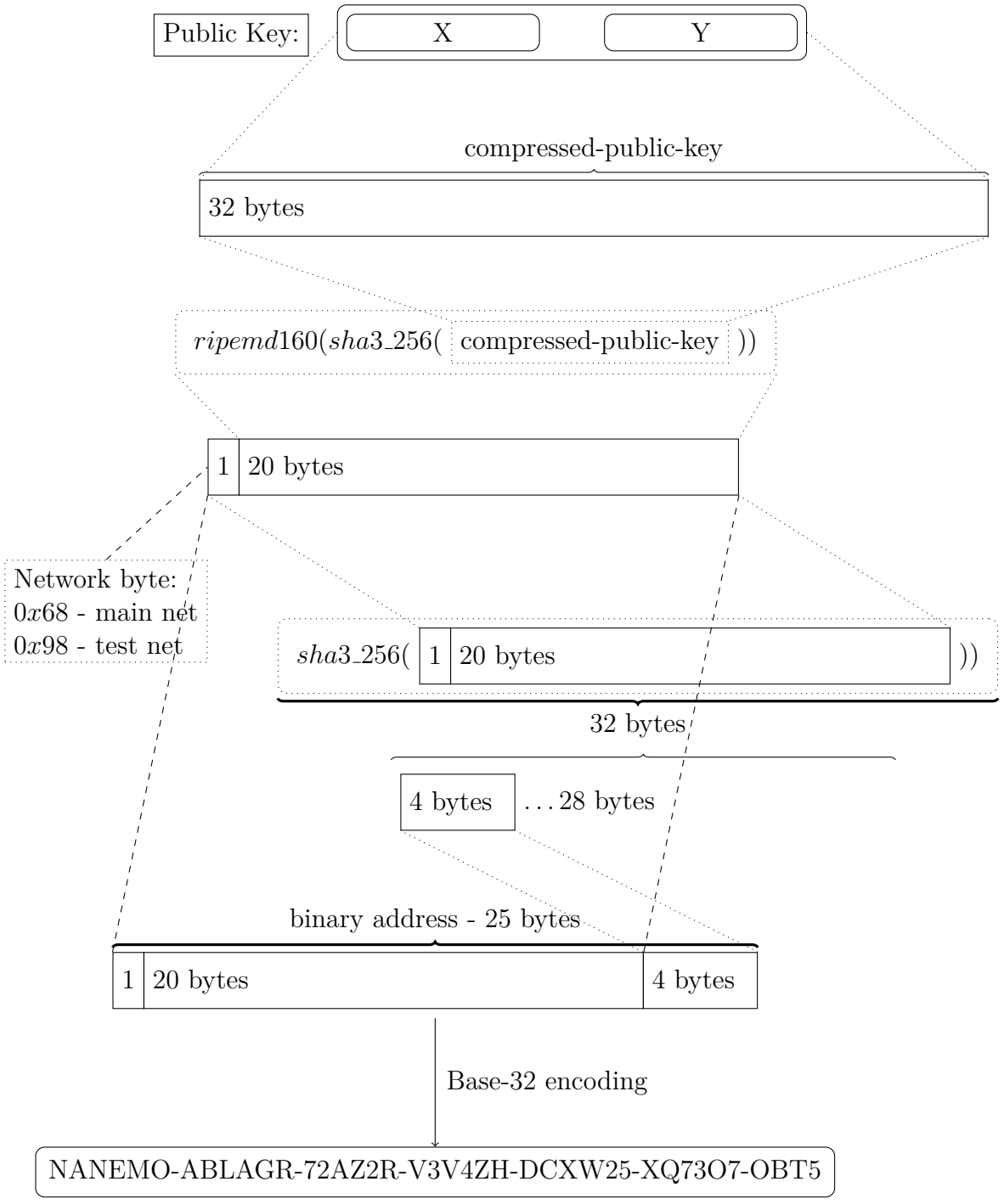


Figure 2: *Address generation*

2.4 Intentional address collision

It is possible that two different public keys will yield the same address. If such an address contains XEM it would be possible for an attacker to withdraw funds from such account.

In order for the attack to succeed, the attacker would need to find a private+public keypair such that the sha3_256 of the public key would **at the same time** be equal to the ripemd-160 preimage of 160-bit hash mentioned above. Since sha3_256 offers 128 bits of security, it's mathematically improbable for a single sha3_256 collision to be found. Due to similarities between NEM addresses and Bitcoin addresses, the probability of causing a NEM address collision is roughly the same as that of causing a Bitcoin address collision.

3 Cryptography

> I understood the importance in principle of public key cryptography but it's all moved much faster than I expected. I did not expect it to be a mainstay of advanced communications technology.
>
> - *Whitfield Diffie*

LOCK chain technology demands the use of some cryptographic concepts. NEM, like many other crypto currencies, is using cryptography based on Elliptic Curve Cryptography. The choice of the underlying curve is important in order to guarantee security and speed.

NEM has chosen to use the *Twisted Edwards curve*:

$$-x^2 + y^2 = 1 - \frac{121665}{121666}x^2y^2$$

over the finite field defined by the prime number $2^{255} - 19$ together with the digital signature algorithm called Ed25519. It was developed by D. J. Bernstein et al. and is one of the safest and fastest digital signature algorithms [2].

The base point for the corresponding group G is called B. The group has $q = 2^{252} - 27742317777372353535851937790883648493$ elements. Every group element A can be encoded into a 256 bit integer $\underline{A}$ which can also be interpreted as 256-bit string and $\underline{A}$ can be decoded to receive A again. For details see [2].

For the hash function H mentioned in the paper, NEM uses the 512 bit SHA3 hash function.

3.1 Private and public key

The *private key* is a random 256-bit integer k. To derive the public key $\underline{A}$ from it, the following steps are taken:

$$H(k) = (h_0, h_1, ..., h_{511}) \tag{1}$$

$$a = 2^{254} + \sum_{3 \leq i \leq 253} 2^i h_i \tag{2}$$

$$A = aB \tag{3}$$

Since A is a group element it can be encoded into a 256-bit integer $\underline{A}$ which serves as the public key.

3.2 Signing and verification of a signature

Given a message M, private key k and its associated public key $\underline{A}$, the following steps are taken to create a signature:

$$H(k) = (h_0, h_1, ..., h_{511}) \tag{4}$$
$$r = H(h_{256}, ..., h_{511}, M) \text{ where the comma means concatenation.} \tag{5}$$
$$R = rB \tag{6}$$
$$S = (r + H(\underline{R}, \underline{A}, M)a) \bmod q \tag{7}$$

Then $(\underline{R}, \underline{S})$ is the *signature* for the message M under the private key k. Note that only signatures where $S < q$ and $S > 0$ are considered as valid **to prevent** the problem of *signature malleability*.

To verify the signature $(\underline{R}, \underline{S})$ for the given message M and public key $\underline{A}$ one checks $S < q$ and $S > 0$ and then calculates

$$\tilde{R} = SB - H(\underline{R}, \underline{A}, M)A$$

and verifies that

$$\tilde{R} = R \tag{8}$$

If S was computed as shown in (7) then

$$SB = rB + (H(\underline{R}, \underline{A}, M)a)B = R + H(\underline{R}, \underline{A}, M)A$$

so (8) will hold.

3.3 Encoding and decoding messages

NEM uses Bouncy Castle's AES block cipher implementation in CBC mode[4] to encrypt and decrypt messages.

If Alice has the private key k_A and wants to encrypt a message for Bob who has the public key $\underline{A_B}$ (with corresponding group element A_B) then the shared secret used when setting up the cipher is calculated as follows:

$$a_A \textit{ is computed from } k_A \textit{ according to } (2)$$
$$salt = 32 \text{ random bytes}$$
$$G = a_A A_B$$
$$\textit{shared secret} = \tilde{H}(\underline{G} \veebar \textit{salt})$$

[4] http://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#CBC

where $\tilde{H}$ is the 256-bit SHA3 hash function.

Another 16 random bytes are used as IV data. Thus, the encrypted message payload consists of

1. the salt
2. the IV data
3. the encrypted message block

Decryption works in a similar manner. Bob has to know Alice's public key $\underline{A_A}$ (and his own private key k_B) and the salt to derive the shared secret:

$$\begin{aligned} a_B \textit{ is computed from } k_B \textit{ according to } (2) \\ G = a_B A_A \\ \textit{shared secret} = \tilde{H}(\underline{G} \veebar \textit{salt}) \end{aligned}$$

Supplying the shared secret and the IV data to the cipher engine decrypts the encoded message.

4 Transactions

> To transact business with the girl who ran the gas-pump Dean merely threw on his T-shirt like a scarf and was curt and abrupt as usual and got back in the car and off we roared again.
>
> - *Jack Kerouac*

RANSACTIONS introduce dynamism into a cryptocurrency system. They are the only way of altering the state of an account. A newly created transaction that has not yet been included in a block is called an *unconfirmed transaction*. Unconfirmed transactions are not guaranteed to be included in any block. As a result, unconfirmed transactions have no effect on the account state. The account state is only updated when a transaction is included in a harvested block and thereby confirmed.

Different types of transactions exist. Each type has a specific purpose, e.g. transfer XEM from one account to another or convert an account to a multisig account. Since transactions consume resources of the p2p network there is a fee for each transaction. The fee depends on the transaction type and other parameters of the transaction.

Transactions have a deadline. If a transaction is not included in a block before its deadline, the transaction is considered expired and gets dropped by the network nodes.

The following sections describe the different transaction types.

4.1 Transfer transactions

A *transfer transaction* is used to transfer XEM from one account to another. A message of at most 1024 bytes can be attached to each transfer transaction. In the case of an encrypted message, only 960 bytes can contain custom data because the salt and the IV data are part of the encrypted message.

Fees for transfer transactions are divided into two parts:

- .05 XEM per 10,000 XEM transferred, capped at 1.25 XEM
- .05 XEM per commenced 32 message bytes (messageLength / 32 + 1).

Both fee parts are added to give the final fee.

4.2 Importance transfer transactions

NEM allows an account to lease its harvesting power to another account through an *importance transfer transaction*. This is known as *delegated harvesting*. This allows the original account to use its importance to harvest on a remote server (such as a virtual private server (VPS)) without needing to have its private key openly exposed on the server. In fact, this feature allows accounts in cold storage to harvest without putting any funds at risk.

The fee for an importance transfer transaction is .15 XEM.

4.2.1 Activating

An account can enable delegated harvesting by sending a special importance transfer transaction that specifies the delegated account.

After the importance transfer transaction is accepted by the network, **360 confirmations** are needed before delegated harvesting activation is complete.

During the activation period, only the original account can harvest but the delegated account cannot. After the activation period, only the delegated account can harvest but the original account cannot.

4.2.2 Deactivating

An account with delegated harvesting activated can disable delegated harvesting at any time by sending a special importance transfer transaction that specifies the delegated account.

After the importance transfer transaction is accepted by the network, **360 confirmations** are needed before delegated harvesting deactivation is complete.

During the deactivation period, only the delegated account can harvest but the original account cannot. After the deactivation period, only the original account can harvest but the delegated account cannot.

4.3 Multisig related transaction types

NEM natively supports m-of-n multisignature accounts.

NEM multisig transactions have been designed with flexibility in mind. Any other

transaction (as of now: importance transfer, transfer, aggregate modification), can be wrapped in a multisig transaction.

A Multisig transaction itself cannot be wrapped inside another *multisig transaction*.

4.3.1 Aggregate modification transactions (multisig modification)

Creating a multisig account An account can be converted into a *multisig account* by sending a special *aggregate modification transaction* that is signed by the multisig account. The transaction specifies information about the cosignatories. The number of cosignatories is limited to 32.

Subsequently, no transactions can be initiated from the multisig account.

Modifying a multisig account After a multisig account has been created, a cosignatory can be added or removed by wrapping an aggregate modification transaction in a multisig transaction. A single modification transaction can add one or more cosignatories but can remove at most one.

- When adding cosignatories, all existing cosignatories must co-sign the transaction.
- When removing a cosignatory, all existing cosignatories except for the one being removed must co-sign the transaction.

The fee for an aggregate modification transaction is a flat fee of 0.5 xem, no matter what modifications you make.

where the term 'modification' refers to the addition or deletion of a cosignatory.

4.3.2 Multisig signature transactions

NEM uses multisig signature transactions for letting cosignatories sign a multisig transaction.

The fee for such a signature transaction is always .15 XEM. The fee is deduced from the multisig account, not from the cosignatory's account.

A multisig signature transaction can only be included in the block chain if the corresponding multisig transaction has been included in the block chain. An orphaned multisig signature transaction will never be included in the block chain.

4.3.3 Multisig transactions

As mentioned earlier, any transaction can be wrapped in a *multisig transaction*.

To send XEM from a *multisig account* to another account, a transfer transaction must be wrapped. The multisig wrapper transaction has a fee of .15 XEM. This fee is added to the usual transfer transaction fee.

The following example shows the steps that must be taken in more detail:
Assume that a multisig account (**M**) has a balance of 1000 XEM and has three cosignatories (**A, B, C**) and **100 XEM** needs to be transferred from **M** to another account **X**.

Any of the three cosignatories can initiate the 100 XEM transfer. Assuming that B initiates the transfer and A and C cosign, the following steps must happen for the transaction to be accepted:

1. B creates a regular, unsigned transfer transaction that has the multisig account as the "signer" and the transfer amount as 100 XEM
2. B wraps the unsigned transfer transaction in a multisig transaction
3. B signs the multisig transaction and sends it to the NEM network
4. A and C are notified of the pending multisig transaction
5. A creates a multisig signature transaction by signing the hash of the unsigned transfer transaction and sends it to the network
6. C creates a multisig signature transaction by signing the hash of the unsigned transfer transaction and sends it to the network
7. Once all cosignatories (B implicitly and A and C explicitly) have signed the unsigned transfer transaction, the transaction is accepted by the network and 100 XEM is transferred from M to X

If A and/or C do not send a multisig signature transaction corresponding to the multisig transfer transaction before the transaction deadline, the multisig transfer transaction will be rejected by the network and no XEM will be transferred from M to X.

4.4 Unconfirmed transactions (spam filter)

When a new transaction is created and delivered to a node, the node

1. Puts the transaction into its unconfirmed transaction cache
2. Broadcasts the transaction to other nodes (if the transaction is valid)

There is also a poll mechanism for unconfirmed transactions during each block chain synchronization round. This allows nodes that do not have their port 7890 open (and therefore cannot receive broadcasts) to learn about new unconfirmed transactions.

Low transaction fees might tempt some bad actor in the network to flood the network with new transactions in order to disturb the network. It is therefore needed to limit the number of unconfirmed transactions which are handled by the nodes.

Simply limiting the number of unconfirmed transactions that a node accepts is bad because normal actors still should be able to send a transaction even when someone is spamming the network. Limiting the number of unconfirmed transactions per account is also not an option since the attacker can create as many accounts as (s)he wants.

NEM does filtering in a smarter way. A custom spam filter decides whether a new unconfirmed transaction should be processed or rejected. It works the following way:

- Consider the unconfirmed transaction cache as having 1000 slots
- As long as less than 120 slots are filled, no transaction is rejected
- Otherwise if there are already *fs* filled slots and a new unconfirmed transaction with signer A arrives, the fair share of slots for account A in the cache is calculated as

$$\textit{eff. importance} = (\textit{importance of } A) + \max\left(0.01, \frac{\textit{transaction fees}}{100000}\right)$$

$$\textit{fair share} = \textit{(eff. importance)} \cdot \exp\left(-\frac{fs}{300}\right) \frac{1000\,(1000 - fs)}{10}$$

  If A occupies less slots than its fair share, the new unconfirmed transaction is accepted.

Note that you can increase the chance that your new transaction is accepted by voluntarily increasing the transaction fees.

Figure 3 shows the fair share of slots versus the fill level of the cache for an effective importance of 1‰. An attacker that tries to occupy many slots cannot gain much by using many accounts as the importance of each account will be very low. He could attack by increasing his transaction fees but that will make him expend his funds at a higher rate.

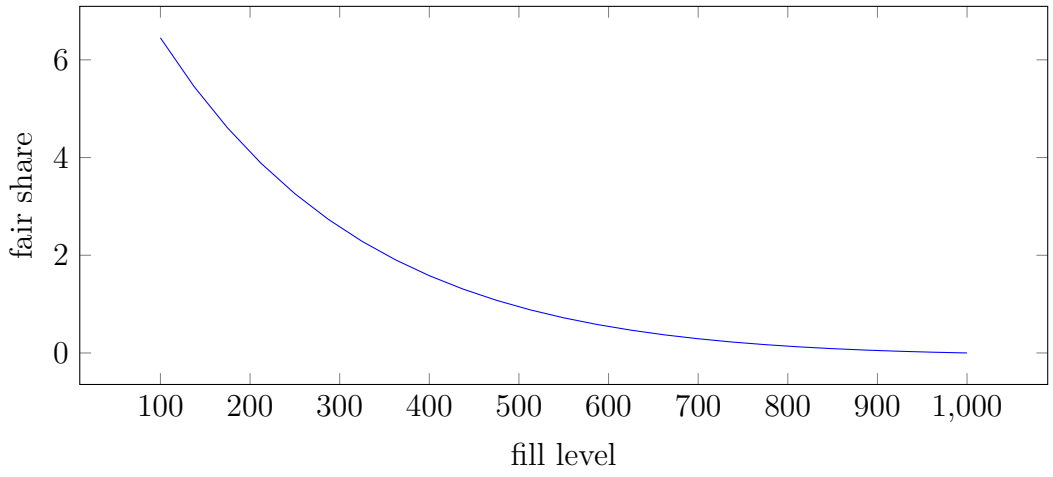


Figure 3: *fair share for effective importance of 1‰*

5 Blocks and the block chain

> If ever I to the moment shall say:
> Beautiful moment, do not pass away!
> Then you may forge your chains to bind me,
>
> - *Johann Wolfgang von Goethe*

The central element of every crypto currency is a public ledger called the block chain that links blocks together. Each NEM block can contain up to 120 transactions. Since the blocks in the chain and the transactions in the blocks are ordered, the complete transaction history is held in the block chain. Blocks are stored in a database as permanent medium. NEM calls the first block in the chain the *nemesis block*.

Each block consists of the following parts:

1. the block version
2. the block time stamp
3. the public key of the harvester (block creator)
4. the signature for the block data
5. the previous block hash
6. the generation hash (needed for block creation)
7. the block height
8. the list of transactions

In the following sections, the hash function H always means the 256-bit Sha3 hash function.

5.1 Block difficulty

The difficulty for a new block is calculated from the difficulties and time stamps of the last 60 blocks. If less than 60 blocks are available, only those are taken into account.

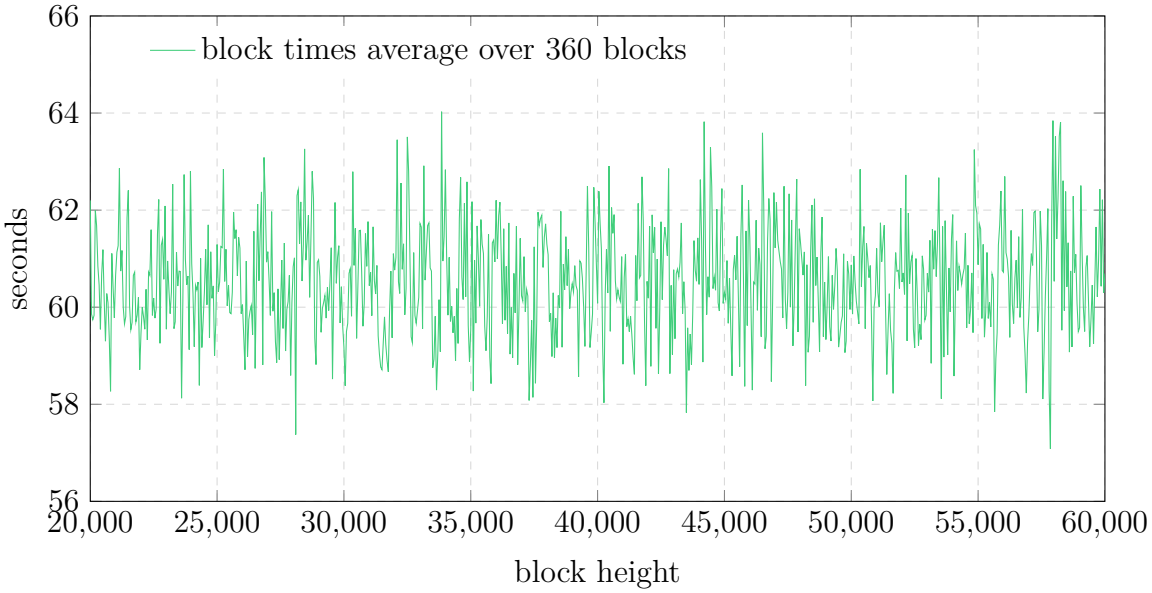


Figure 4: *Main net average block times over 360 blocks*

If only one block is available, then the block has a predefined *initial difficulty* of 10^{14}. Otherwise, the difficulty is calculated from the last n blocks the following way:

$$d = \frac{1}{n}\sum_{i=1}^{n}(\text{difficulty of block i}) \qquad \text{(average difficulty)}$$

$$t = \frac{1}{n}\sum_{i=1}^{n}(\text{time to create block i}) \qquad \text{(average creation time)}$$

$$\textit{difficulty} = d\,\frac{60}{t} \qquad \text{(new difficulty)}$$

If the new difficulty is more than 5% greater or smaller than the difficulty of the last block, then the change is capped to 5%.

Additionally, difficulties are kept within certain bounds. The new difficulty is clamped to the boundaries if it is greater than 10^{15} or smaller than 10^{13}.

Simulations and the NEM beta phase have shown that the algorithm produces blocks with an average time of 60 ± 0.5 seconds.

The slow change rate of 5% makes it hard for an attacker with considerably less than 50% importance to create a better chain in secret since block times will be considerably higher than 60 seconds for the beginning of his secret chain.

5.2 Block score

The score for a block is derived from its difficulty and the time (in seconds) that has elapsed since the last block:

$$score = difficulty - time\ elasped\ since\ last\ block \qquad \text{(block score)}$$

5.3 Block creation

The process of creating new blocks is called *harvesting*. The harvesting account gets the fees for the transactions in the block. This gives the harvester an incentive to add as many transactions to the block as possible. Any account that has a *vested balance* of at least 10,000 XEM is eligible to harvest.

To check if an account is allowed to create a new block at a specific network time, the following variables are calculated:

$$\begin{aligned} h &= H(generation\ hash\ of\ previous\ block,\ public\ key\ of\ account) \\ &\quad \text{interpreted as 256-bit integer} \\ t &= time\ in\ seconds\ since\ last\ block \\ b &= 8999999999 \cdot (importance\ of\ the\ account) \\ d &= difficulty\ for\ new\ block \end{aligned}$$

and from that the *hit* and *target* integer values:

$$\begin{aligned} hit &= 2^{54} \left| \ln\left(\frac{h}{2^{256}}\right) \right| \\ target &= 2^{64}\ \frac{b}{d}\ t \end{aligned}$$

The account is allowed to create the new block whenever $hit < target$. In the case of delegated harvesting, the importance of the original account is used instead of the importance of the delegated account.

Since *target* is proportional to the elapsed time, a new block will be created after a certain amount of time even if all accounts are unlucky and generate a very high hit.

Also note that *hit* has an exponential distribution. Therefore, the probability to create a new block does not change if the importance is split among many accounts.

5.4 Block chain synchronization

Since blocks are assigned a score, a score can be assigned to a chain of blocks too:

$$score = \sum_{block \in blocks} block\ score \qquad \text{(block chain score)}$$

Block chain synchronization is a central task for every block chain based crypto currency. From time to time a local node will ask a remote node about its chain. The remote node is selected using the calculated trust values (see section 6: A reputation system for nodes).

If the remote node promises a chain with a higher score, the two nodes try to agree on the last common block. If successful, the remote node will supply up to 400 blocks of its chain to the local node.

If the supplied chain is valid, the local node will replace its own chain with the remote chain. If the supplied chain is invalid, the local node will reject the chain and consider the synchronization attempt with the remote node to have failed.

This algorithm will also resolve forks. The last common block may have a height difference of at most 360 compared to the local node's last block. Thus, the maximal depth of forks that can be resolved via the synchronization algorithm is 360.

The flow chart on the next page illustrates the process in more detail.

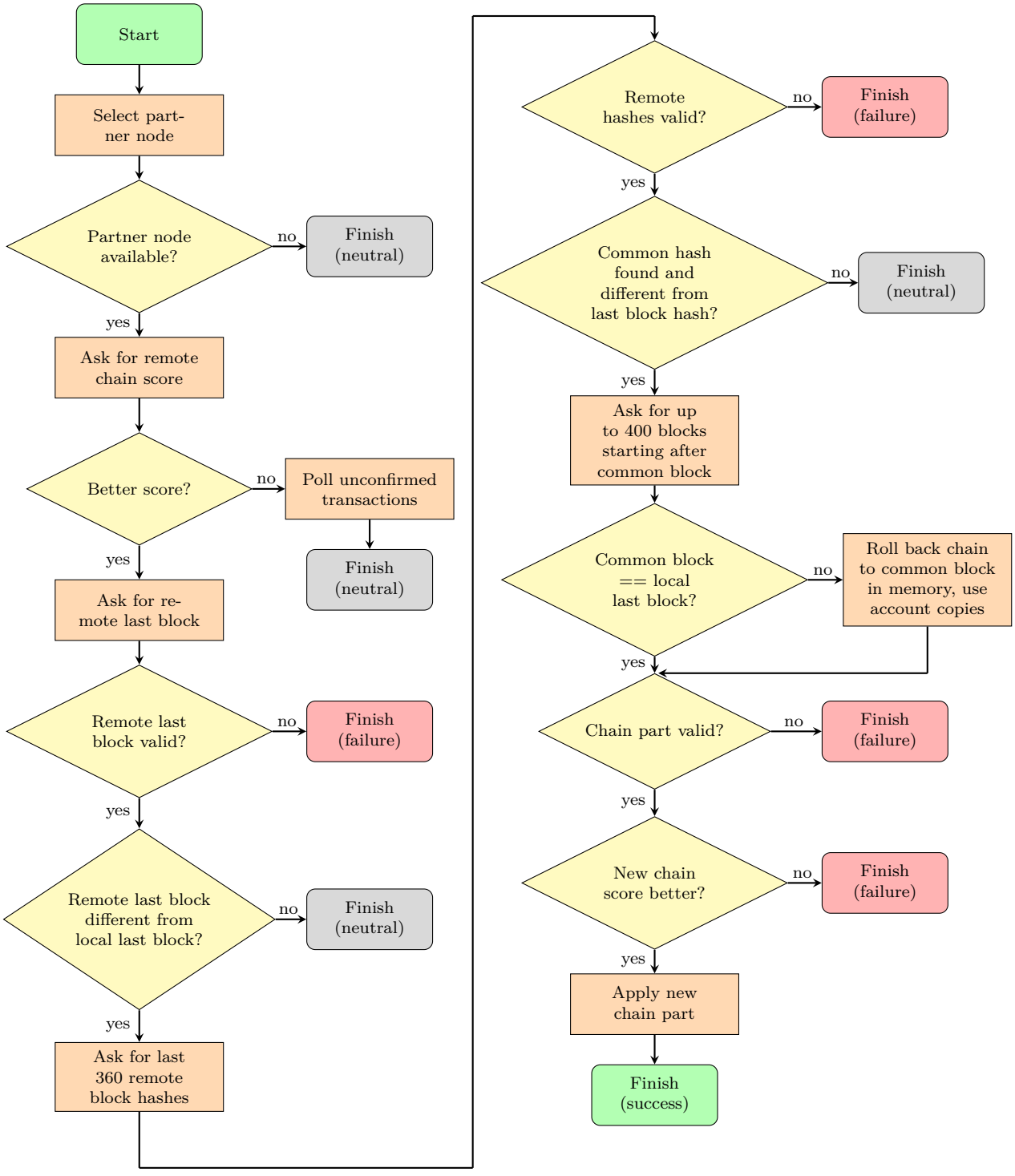


Figure 5: *Block chain synchronization flow chart*

6 A reputation system for nodes

> Beauty is something that burns the hand when you touch it.
>
> - *Yukio Mishima*

Like other crypto currency networks, the NEM network is a peer-to-peer (P2P) network. P2P networks have the great advantage of being robust because they cannot be shut down by eliminating a single entity. Nevertheless, experience in the last few years has shown that P2P networks also have their share of disadvantages. The participants of the network are anonymous and anyone can join. This makes it very easy to inject hostile nodes into the network that spread invalid information or try to disturb the network in some way.

There is a need to identify hostile nodes and reduce communication with them. There have been many approaches to achieve this. One of the most successful is building a reputation system for nodes. NEM follows this approach by implementing an algorithm similar to the EigenTrust++ reputation system. This section will outline the algorithm used.

For a more detailed discussion, see the original papers about EigenTrust[8] and EigenTrust++[5].

6.1 Node interactions

Nodes in the NEM network interact with each other by sharing information about entities like transactions and blocks of transactions. A node can either broadcast new entities to other nodes or ask other nodes for those entities.

Having received information from a remote node, a node can verify the validity of the information and check if the information is usable. Each node can decide if an interaction (or 'call') should be seen as a success (new valid information was received), neutral (valid, but already known, information was received) or failure (invalid information was received).

Each node i keeps track of the outcomes of all of its interactions with node j in its experience map by counting its successful and failed interactions; $\mathit{success}(i, j)$ and $\mathit{failed}(i, j)$. Neutral interactions are ignored.

6.2 Local trust value

Each node starts with a set P of so called pre-trusted nodes that are supplied in a configuration file and can be edited by the user to his/her own needs. The node then asks

the pre-trusted nodes for other existing nodes in the network. Knowing about n nodes in the network each node can build an initial trust vector $\vec{p}$ by setting:

$$p_j = \begin{cases} \frac{1}{|P|} & \text{if } node_j \in P \\ 0 & \text{otherwise} \end{cases}$$

p_j indicates how much trust a node initially has in node j.

After some time node i had some interactions with other nodes and can update its local trust values by first calculating:

$$s_{ij} = \begin{cases} \frac{success(i,j)}{success(i,j)+failed(i,j)} & \text{if } success(i,j) + failed(i,j) > 0 \\ p_j & \text{otherwise} \end{cases}$$

and then normalizing the local trust values:

$$c_{ij} = \frac{s_{ij}}{\sum_m s_{im}}$$

to define the local trust vector $\vec{c_i}$ with components c_{ij}.

6.3 Aggregating local trust values

From time to time nodes broadcast their local trust values to other nodes. Having received the local trust values from other nodes, node i can calculate an aggregate trust value for node k by weighing the local trust node j has in node k with its own trust in node j:

$$t_{ik} = \sum_j c_{ij} c_{jk}$$

This can be written in matrix notation by defining $C = (c_{kj})$ and $\vec{t_i}$ having components t_{ik}:

$$\vec{t_i} = C^T \vec{c_i}$$

If we define the iteration:

$$\vec{t}_{i+1} = C^T \vec{t_i}$$

then this will converge to the left principal eigenvector $\vec{t}$ of the matrix C under the assumptions that C is irreducible and aperiodic. To guarantee the assumptions being valid, we slightly change the iteration by mixing a portion of the initial trust vector $\vec{p}$ into the iteration:

$$\vec{t_i} = \begin{cases} \vec{p} & \text{if } i = 0 \\ (1-a)C^T \vec{t}_{i-1} + a\vec{p} & \text{otherwise} \end{cases} \tag{9}$$

where a suitable $0 < a < 1$ is chosen. This iteration will always converge to a vector $\vec{t}$, which represents the trust a node has in other nodes.

6.4 Enhancing the algorithm

The above algorithm to compute trust suffers from the fact that the veracity of the feedback reported from other nodes is unknown. Malicious nodes could collude and report low trust values for honest nodes and high trust values for dishonest nodes.

An improvement is to estimate the credibility of the feedback of other nodes and weight the reported trust values by the credibility score. To do that, $common(u, v)$ is defined for two nodes u and v as the set of nodes with which both nodes have interacted. Given that, a measure for the similarity of the feedback of the two nodes ($sim(u, v)$) can be calculated:

$$sim(u,v) = \begin{cases} \left(1 - \sqrt{\frac{\sum_{w \in common(u,v)} (s_{uw} - s_{vw})^2}{|common(u,v)|}}\right)^b & \text{if } common(u,v) \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

where b is a positive integer (the original paper suggests $b = 1$).

Then, feedback credibility can be defined as:

$$f_{ij} = \begin{cases} \frac{sim(i,j)}{\sum_m sim(i,m)} & \text{if } \sum_m sim(i,m) > 0 \\ 0 & \text{otherwise} \end{cases}$$

and, finally, the matrix $L = (l_{ij})$ can be defined as:

$$l_{ij} = \begin{cases} \frac{f_{ij} c_{ij}}{\sum_m f_{im} c_{im}} & \text{if } \sum_m f_{im} c_{im} > 0 \\ 0 & \text{otherwise} \end{cases}$$

which incorporates both the reported trust values and the feedback credibility.

It is now straightforward to define an iteration analog to Equation 9:

$$\vec{t_i} = \begin{cases} \vec{p} & \text{if } i = 0 \\ (1-a) L^T \vec{t}_{i-1} + a\vec{p} & \text{otherwise} \end{cases}$$

which converges to the left principal eigenvector of the underlying matrix of the power iteration.

The original Eigentrust++ paper suggests using additional measures to limit trust propagation between honest and dishonest nodes. The paper was written with file sharing networks in mind. In such networks, incomplete data is shared (parts of files) and cannot be checked for validity. Therefore, even honest nodes could distribute malicious data.
Nodes in the NEM network always download entities in their entirety and verify their integrity before distributing them to other nodes. NEM network simulations have shown that the results without the additional trust propagation measures are good enough to mitigate the presence of malicious nodes.

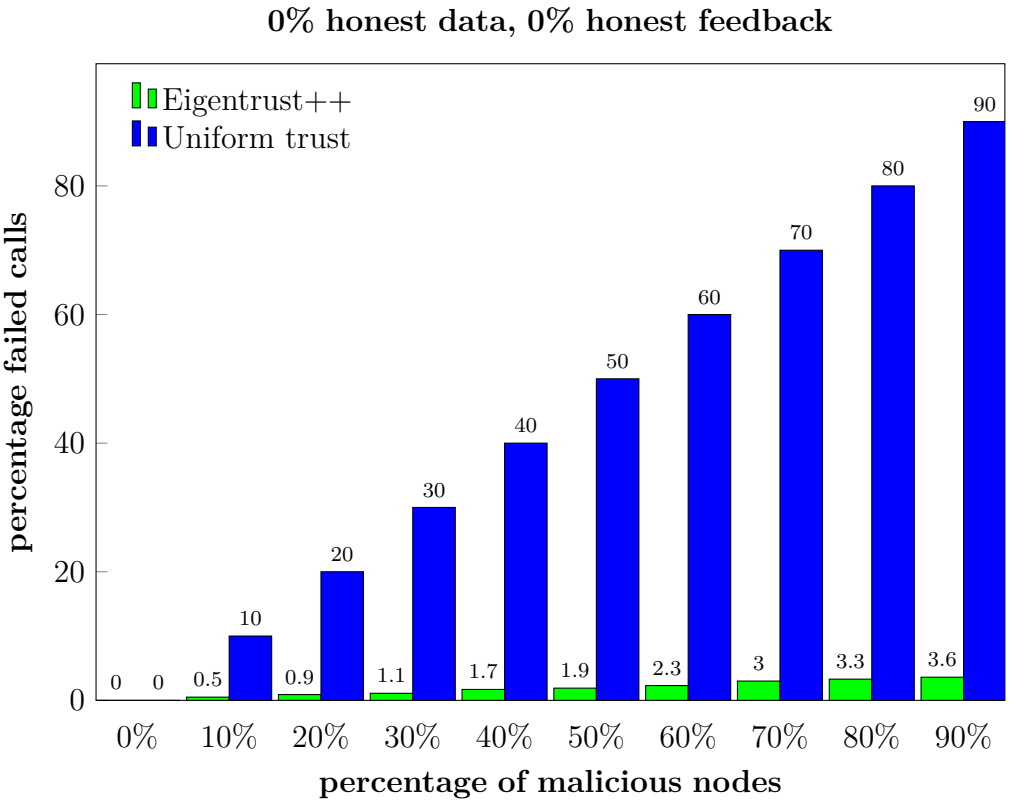


Figure 6: *Simulation with attacking nodes that are always dishonest*

6.5 Benefits of the reputation system

Having a reputation system for nodes allows nodes to select their communication partner according to the trust values for other nodes. This should also help balance the load of the network because the selection of the communication partner only depends on a node's honesty but not its importance.

Simulations show that the algorithm reduces the number of failed interactions considerably. If malicious nodes only provide dishonest data and dishonest feedback they are easily identified (Figure 6).

But even if the malicious nodes collude to give other malicious nodes a high trust value and provide false data and feedback only to a certain percentage, the trust algorithm still cuts down the percentage of failed interactions (Figure 7).

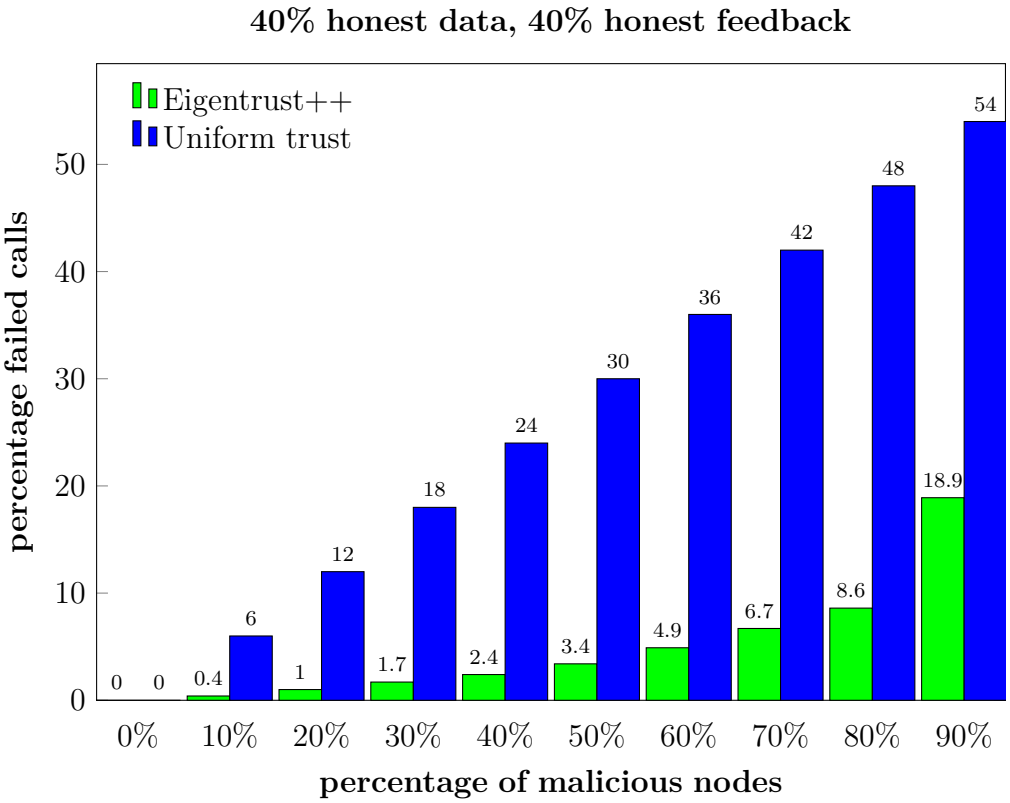


Figure 7: *Simulation with attacking nodes that are sometimes dishonest*

7 Proof-of-Importance

> It was hot, the night we burned Chrome.
>
> - *William Gibson*

Proof-of-Importance (PoI) is the name of the block chain consensus algorithm used by NEM. Each account is assigned an importance score that proxies its aggregate importance to the NEM economy. Accounts with higher importance scores have higher probabilities of harvesting a block (see section 5: Blocks and the block chain). Because all transactions are publicly available in NEM, the transaction graph of the NEM economy can be calculated exactly. The topology of the transaction graph can be used as an input into the importance of an account. The insight that the transaction graph can be used for elucidating the importance of an account is the key innovation of Proof-of-Importance.

The NEM block chain platform allows all transactions to be transparently viewed. This information about value transfers between accounts can be used to determine a rating for the importance of accounts. The intuition that not all nodes in a graph have the same salience, or importance, is not new. The literature in the graph theory community is well established for computing the importance of nodes in graphs in the areas of search [11], social networks [1], street networks [7], and neuroscience [6], among others. Building off of this intuition, one of NEM's core innovations is to use graph theoretic measures as a fundamental input into block chain consensus. The outlink matrix that defines the transaction graph is centrally important and used in the PoI calculation.

7.1 Eligibility for Entering the Importance Calculation

To be eligible for entering the importance calculation, an account must have at least 10,000 vested XEM. All accounts owning more than 10,000 vested XEM have a non-zero importance score.

With a supply of 8,999,999,999 XEM, the theoretical maximum number of accounts with non-zero importance is 899,999. In practice, the number of actual accounts with non-zero importance is not expected to approach the theoretical max due to inequalities in held XEM and also the temporal costs associated with vesting.

If NEM becomes very popular, a threshold of 10,000 vested XEM could be undesirable. If necessary, this number could be updated in the future via a hard fork, which is the same procedure for adjusting transaction fees and other parameters related to harvesting.

7.2 The outlink matrix

Suppose the PoI calculation is done at height h. Accounts that have a vested balance of at least 10,000 XEM at height h are eligible to be part of the PoI calculation. For those accounts, NEM collects all transfer transactions that satisfy the following conditions:

- Transferred an amount of at least 1,000 XEM
- Happened within the last 43,200 blocks (approximately 30 days)
- Recipient is eligible to be part of the PoI calculation too (see subsection 7.1: Eligibility for Entering the Importance Calculation)

For each such transaction T_k that transferred *amount* μXEM from account A_i to account A_j and happened at height h_{ijk}, a weight is calculated according to the following formula:

$$w_{ijk} = amount \cdot \exp\left(\ln(0.9)\left[\frac{h - h_{ijk}}{1440}\right]\right)$$

where $[x]$ denotes the floor function. The graph in Figure 8: amount decay of 10000 XEM illustrates how a transaction amount of 10,000 XEM is weighted over time.

These values are aggregated to

$$\widetilde{w}_{ij} = \sum_k w_{ijk}$$

Setting

$$\widetilde{o}_{ij} = \begin{cases} \widetilde{w}_{ji} - \widetilde{w}_{ij} & \text{if } \widetilde{w}_{ji} - \widetilde{w}_{ij} > 0 \\ 0 & \text{otherwise} \end{cases}$$

finally, the outlink matrix $\mathbf{O}$ is comprised of components o_{ij} as

$$o_{ij} = \begin{cases} \frac{\widetilde{o}_{ij}}{\sum_i \widetilde{o}_{ij}} & \text{if } \sum_i \widetilde{o}_{ij} > 0 \\ 0 & \text{otherwise} \end{cases}$$

The outlink matrix element o_{ij} thus describes the **weighted net flow** (which is set to 0 if negative) of XEM from A_i to A_j during the (approximately) last 30 days. This means that only **net** transfers contribute to an account's importance.

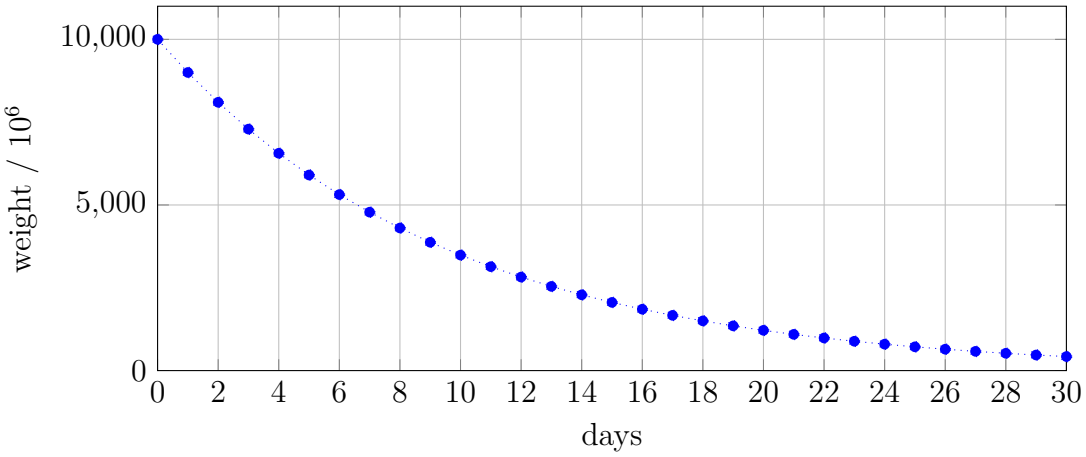


Figure 8: *amount decay of 10000 XEM*

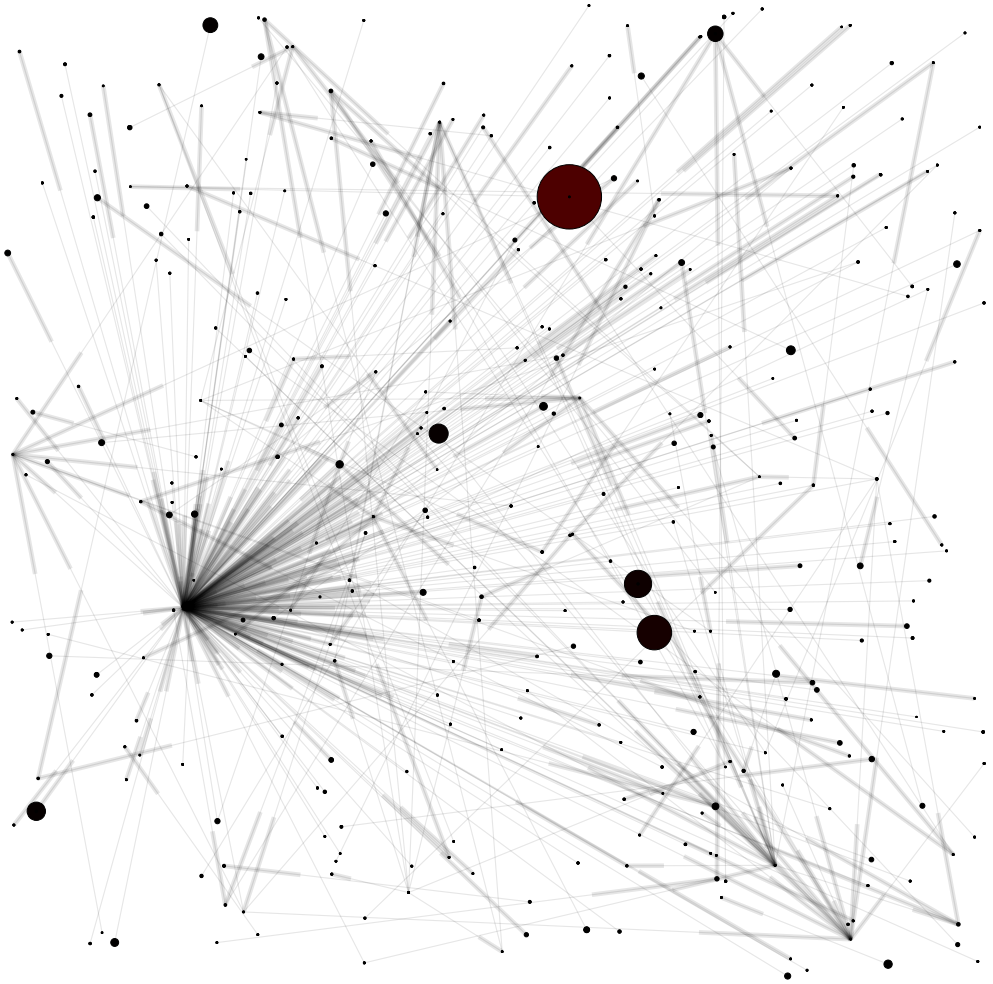


Figure 9: *Plot of the NEM transaction graph (outlink matrix) as of 29 April, 2015, containing 1,456 harvesting-eligible accounts.*

7.3 NCDawareRank

There are many ways to determine the salience of nodes in a network, and PageRank is one method. NCDawareRank is similar to PageRank, where the stationary probability distribution of an ergodic Markov chain is calculated [9, 11]. NCDawareRank additionally exploits the nearly completely decomposable structure of large-scale graphs of information flows by adding an inter-level proximity matrix as an extra term, **M**. The inter-level proximity matrix models the fact that groups of nodes are closely linked together to form clusters that interact with each other. This allows NCDawareRank to converge faster than PageRank while also being more resistant to manipulation of scores, because the rank for nodes within the same level will be limited.

Shown in matrix notation, NCDawareRank is calculated as:

$$\hat{\pi} = \mathbf{O}\eta\pi + \mathbf{M}\mu\pi + \mathbf{E}(1 - \eta - \mu)\pi, \tag{10}$$

where:

$\mathbf{O}$ is the outlink matrix
$\mathbf{M}$ is the inter-level proximity matrix
$\mathbf{E}$ is the teleportation matrix
π is the NCDawareRank
η is the fraction of importance that is given via outlinks
μ is the fraction of importance given to proximal accounts

This definition is the same as for PageRank, only with the addition of $\mathbf{M}$ and μ. For NEM, η is 0.7 and μ is 0.1. The details of how each of these variables is calculated are as follows.

Let W be the set of all harvesting-eligible accounts. For $u \in W$, G_u is the set of accounts that have received more in value transfers from account u than have sent u. Nearly completely decomposable (NCD) partitions of W are defined as $\{A_1, A_2, ..., A_N\}$, such that for every $u \in W$, there is a unique K such that $u \in A_K$. The proximal accounts of each u, χ_u, are thus defined as:

$$\chi_u \triangleq \bigcup_{w \in (u \cup G_u)} A_{(w)}, \tag{11}$$

and N_u denotes the number of NCD blocks in χ_u.

The definition of the outlink matrix $\mathbf{O}$ is described in subsection 7.2: The outlink matrix. To guarantee convergence, $\mathbf{O}\eta + \mathbf{M}\mu + \mathbf{E}(1-\eta-\mu)$ must be an irreducible, column-stochastic matrix. This is accomplished by dispersing the rank of dangling accounts (accounts without outlinking value transfers) so that every account has a non-zero teleportation probability.

The inter-level proximity matrix $\mathbf{M}$ is calculated by clustering the transaction graph (described in subsection 7.4: Clustering the transaction graph) to define each NCD block A_N and then determining the proximal accounts for each cell vu in $\mathbf{M}$:

$$M_{v,u} \triangleq \begin{cases} \frac{1}{N_u \left| A_{(v)} \right|} & \text{if } v \in \chi_u \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

The teleportation matrix $\mathbf{E}$ is calculated as:

$$\mathbf{E} \triangleq \mathbf{e}\mathbf{v}^{\top} \tag{13}$$

where $\mathbf{v}^{\top}$ is a teleportation probability vector and $\mathbf{e}$ is the vector with all components set to 1.

Practically, NCDawareRank is calculated via the power iteration method as follows:

$$\begin{aligned} NCDawareRank^{r}(i) = (1-\eta-\mu)\frac{1}{|G|} + \\ \eta \sum_{k=1}^{s} o_{ik}\ NCDawareRank^{r-1}(k) + \\ \mu \sum_{k=1}^{s} m_{ik}\ NCDawareRank^{r-1}(k) \end{aligned} \tag{14}$$

where o_{ki} denotes the k^{th} row for column i in $\mathbf{O}$ and m_{ki} is the k^{th} row for column i in $\mathbf{M}$. The algorithm continues until the change in NCDawareRank between iterations is less than a specified ε:

$$\left(\sum_{i \epsilon G} \left| NCDawareRank^{r}(i) - NCDawareRank^{r-1}(i) \right| \right) < \varepsilon \tag{15}$$

Teleportation for both the outlink and the inter-level proximity matrices ensures that the transition probability matrix between accounts is stochastic, irreducible, and primitive, so that the algorithm is guaranteed to converge. For more details on the mathematical theory of PageRank, see [9].

As discussed in [10], it is possible to reduce the number of calculations and speed up the calculation of NCDawareRank by decomposing the sparse inter-level proximity matrix $\mathbf{M}$ into two matrices $\mathbf{R}$ and $\mathbf{A}$:

$$\mathbf{A} \triangleq \begin{bmatrix} e_{|A_1|} & 0 & \cdots & 0 \\ 0 & e_{|A_2|} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & e_{|A_N|} \end{bmatrix} \tag{16}$$

$e_{|A_k|}$ is the vector with $|A_k|$ components all set to 1.

$$c = (\frac{1}{|A_1|} \quad \frac{1}{|A_2|} \quad \cdots \quad \frac{1}{|A_N|}) \tag{17}$$

$$\mathbf{R} \triangleq \begin{bmatrix} c_1\mathbf{R}'_{1*} & c_2\mathbf{R}'_{2*} & \cdots & c_N\mathbf{R}'_{\mathrm{N}*} \end{bmatrix} \tag{18}$$

where $\mathbf{R}'_{i*}$ is the i^{th} row of the matrix $\mathbf{R}'$ which is defined as

$$R_{i,j} \triangleq \begin{cases} \frac{1}{N_u} & \text{if } A_i \in \chi_u \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

The NEM implementation uses the decomposition of $\mathbf{M}$ into $\mathbf{A}$ and $\mathbf{R}$. See [10] for a thorough discussion of the storage and computational savings of this decomposition.

7.4 Clustering the transaction graph

Clustering is done on the transaction graph using a high-performance implementation [13][5] of the SCAN clustering algorithm [14]. In this high-performance implementation, clusters are created by finding nodes that are *core* and then expanding the clusters, calculating the structural similarity between groups of nodes that are two-hops away from each other. Details of the algorithm are as follows.

Assume a graph G of accounts V, where edges E between accounts are defined such that an edge is created if the sum of the decayed value transfers (see subsection 7.2: The outlink matrix) between the accounts is over a predefined threshold of 1,000 XEM. Clustering of accounts in the transaction graph G is done by performing structural similarity-based clustering, which elucidates clusters, hubs, and outliers in the graph. The structural

[5] http://db-event.jpn.org/deim2014/final/proceedings/D6-2.pdf

similarity between two accounts u and v in the transaction graph, $\sigma(u, v)$, is calculated as follows:

$$\sigma(u, v) = \frac{|\Gamma(u) \cap \Gamma(v)|}{\sqrt{|\Gamma(u)|\,|\Gamma(v)|}}, \tag{20}$$

where $|.|$ denotes set cardinality and Γ is the set of structurally connected accounts (inclusive of self), defined as:

$$\Gamma(u) = \{v \in V | \{u, v\} \in E\} \cup \{u\}\,. \tag{21}$$

N_ϵ is the set of structurally connected accounts that have structural similarity with an account over a pre-determined threshold, ϵ:

$$N_\epsilon(u) = \{v \in \Gamma(u) | \sigma(u, v) \geq \epsilon\}\,. \tag{22}$$

Core nodes are used for pivoting and expanding clusters and are defined as follows:

$$K_{\epsilon,\mu}(u) \Leftrightarrow |N_\epsilon(u)| \geq \mu \tag{23}$$

where μ is the minimum number of epsilon neighbor accounts that an account must have to be considered *core*. During clustering, clusters are centered (pivoted) around *core* accounts. The initial members of the cluster are the members of N_ϵ. This means that μ controls the smallest possible size of a cluster. In NEM, μ is 4 and ϵ is 0.3. An account v has *direct structure reachability*, $u \mapsto_{\epsilon,\mu} v$, with account u for a given ϵ and μ, if u is *core* and v is a member of $N_\epsilon(u)$:

$$u \mapsto_{\epsilon,\mu} v \Leftrightarrow K_{\epsilon,\mu}(u) \wedge v \in N_\epsilon(u). \tag{24}$$

In the SCAN algorithm, accounts that are *core* are set as pivots and then clusters are expanded by including accounts with direct structure reachability (Equation 24: Clustering the transaction graph). This requires computing the structural similarity with each neighbor and the neighbors' neighbors.

The improved version of SCAN only looks at the pivot accounts and accounts that are two-hops away from the pivot accounts. Accounts two-hops away from account u, $H(u)$, are defined as follows:

$$H(u) = \{v \in V | (u, v) \notin E \wedge (v, w) \in E)\}\,, \tag{25}$$

where account w is an account, such that $w \in N_\epsilon(u) \setminus \{u\}$. For each *core* account that is two-hops away from the pivot, a new cluster is generated and pivoted around it. All of the *core* account's epsilon neighbors (N_ϵ) are added to the new cluster. When computing the accounts that are two-hops away, accounts with *direct structure reachability* from the pivoted node are removed from the calculation. When expanding the two-hops away accounts, the accounts are processed, such that:

$$H(u_n) = \left\{ v \in V | (u,v) \notin E \wedge (v,w) \in E \wedge v \notin \bigcup_{i=0}^{n-1} N_\epsilon(u_i) \cup H(u_i) \right\}. \tag{26}$$

After all accounts in the graph have been processed, all nodes are analyzed. If an account belongs to multiple clusters, then those clusters are merged. Afterwards, any account that is not in a cluster is marked as a hub if it connects two or more clusters or as an outlier if it does not.

The use of the two-hop away nodes to expand the clusters reduces the computation cost of clustering because the calculation of structural similarity σ is the slowest part of the algorithm.

The computed clusters are also used to determine the levels in the NCDawareRank inter-level proximity matrix, as these clusters are representative of the nearly completely decomposable nature of the transaction graph.

7.5 Calculating Importance Scores

The importance score, ψ, is calculated as follows:

$$\psi = (\text{normalize}_1(max(0, \nu + \sigma w_o)) + \hat{\pi} w_i)\chi, \tag{27}$$

where:

normalize$_1(v)$ is: $\dfrac{v}{\|v\|}$

ν is the vested amount of XEM

σ is the weighted, net outlinking XEM

$\hat{\pi}$ is the NCDawareRank [10] score

χ is a weighting vector that considers the structural topology of the graph

w_o, w_i are suitable constants

χ considers the topology of the graph and assigns a higher weight to nodes that are members of clusters, rather than outliers or hubs. Outliers and hubs are weighted at 0.9 of their score, whereas nodes that are in clusters are weighted at 1.0. In NEM, w_o is 1.25 and w_i is 0.1337.

Taken together, the information about vested balance, sent XEM, and transaction graph-topology form the basis for heuristic evaluation of the importance of accounts in the NEM economy. Furthermore, since importance cannot be arbitrarily manipulated or gamed (see subsection 7.6: Resistance to Manipulation), importance scores are useful for purposes other than just block chain consensus. For example, they can be interpreted as a form of reputation. Because all importance scores sum to unity, they represent a finite quantity that can be used for purposes such as voting or preventing spam. This allows even anonymous actors to interact with each other because it is not possible for people to gain control by creating many pseudonymous identities.

7.6 Resistance to Manipulation

The use of NCDawareRank, vested balance, decayed and weighted outlinks, and summation to unity in the calculation of importance scores makes the scores resistant to arbitrary manipulation.

7.6.1 Sybil Attack

In peer-to-peer systems, faulty or malicious entities can often present themselves as multiple identities in order to gain control of a system [3]. This is called a Sybil attack.

In NEM, there is a financial reward for harvesting blocks because harvesters receive fees (see section subsection 5.3: Block creation), and accounts with higher importance scores have higher probabilities of harvesting a block. As a result, attackers are expected to be highly motivated to attempt Sybil attacks. Sybil-style attacks were considered when designing the PoI algorithm. The usage of NCDawareRank, vested balances, and net decaying outlink weights makes the importance score calculation resistant to Sybil attacks.

With respect to the importance score calculation, some of the potential vectors for Sybil attackers include:

- account splitting and transacting among split accounts to boost the NCDawareRank score
- account splitting and transacting with random accounts

- sending XEM between acounts in a loop to boost the outlink score (see subsubsection 7.6.2: Loop Attack)

The following mitigations are in place to counter Sybil attacks.

- **Use NCDawareRank instead of PageRank as a graph-theoretic importance metric**
  The inter-level proximity matrix **M** in the NCDawareRank algorithm makes the algorithm more resistant to link spamming than PageRank [10]. For web pages, PageRank is commonly spammed by creating many sites that all link to a main site, magnifying the PageRank of the main site [4]. The analog to this in NEM would be for an account to send portions of its balance to other accounts and then send all the XEM back to the main account.
- **Vest the balance of accounts over a temporal schedule that is qualitatively "slow" for humans**
  Requiring several weeks to fully vest inflows into an account makes it impossible to acquire large amounts of XEM and then immediately attack the network.
- **Use net-outlinking XEM for outlink score calculation**
  The net-outlinking XEM is used in the importance score calculation. There is no advantage in sending XEM around to many accounts because the inlinking XEM sent to the account will cancel out the outlinking XEM.
- **Decay the weight of outlinking value transfers**
  Outlinking value transfers are decayed over time. Sending XEM to another account will only give a temporary boost in the importance score.
- **Normalize importance scores to sum to unity**
  The actions of others affect your importance score because all scores sum to unity.
- **Require a minimum balance of 10,000 vested XEM required for harvesting**
  Requiring at least 10,000 vested XEM to harvest creates bounds on the number of accounts that could theoretically be involved in Sybil attacks.
- **Choose relatively small values for w_o and w_i**
  This ensures that the largest component of importance is the amount of vested XEM, which cannot be exploited.

Taken together, these countermeasures make Proof-of-Importance resistant to Sybil-style attacks. To test this, we performed simulations with between 1 and 10,000 colluding Sybil accounts. Figure 10 shows the results of the simulation for 3 conditions:

- colluding accounts sending XEM in a loop to each other
- colluding accounts all sending XEM to a common, colluding account
- colluding accounts sending XEM to accounts randomly

As figure 10 shows, after adding a small number of accounts, a Sybil attacker does not gain much importance by adding even thousands of additional accounts. Although the Sybil attacker gains more importance than the honest actor, the gain is limited and bounded by the PoI calculation. Some actors can boost their importance by splitting their accounts or or sending transactions to other accounts to mimic economic activity. Because PoI scores sum to unity, other rational actors should take the same actions. This would cause the expected benefit to disappear.

Compared to PoW mining, the advantage that can be gained by gaming PoI is minimal. In PoW, miners who buy specialized mining hardware have a large advantage over those who use only commonly available video cards. As of this writing, an equal amount of money can buy either (1) two video cards with combined 800 Mhash/s of mining power or (2) a specialized ASIC miner with 800,000 Mhash/s. The difference between the mining power of (1) and (2) is approximately 1000x. In PoI, due to the constraints of the algorithm, an actor perfectly gaming the system will get an advantage well less than an order of magnitude. In other words, PoI gaming will not result in a significant qualitative advantage over actors with the same balance who do nothing.

For the case where controlled accounts all send to a common master account, figure 10b shows that the importance can actually decrease when more nodes are involved in an attack. For figure 10b, the importances decreased after a threshold of accounts were combined in the attack because the graph topology changed such that the attacking nodes were no longer in the same cluster. This caused the structural topology weight, χ, to be reduced from 1.0 to 0.9.

7.6.2 Loop Attack

In the loop attack, accounts controlled by the same entity send XEM around via transfer transactions in a loop to boost their importance score. While the outlinking XEM from an account is a large portion of the importance score calculation, the outlinking XEM is the *net* outlinking XEM, such that inlinking XEM to an account is subtracting from the outlinking XEM. Thus sending 1,000 XEM around in a loop millions of times will not give you a higher net outlinking XEM score than sending out just once.

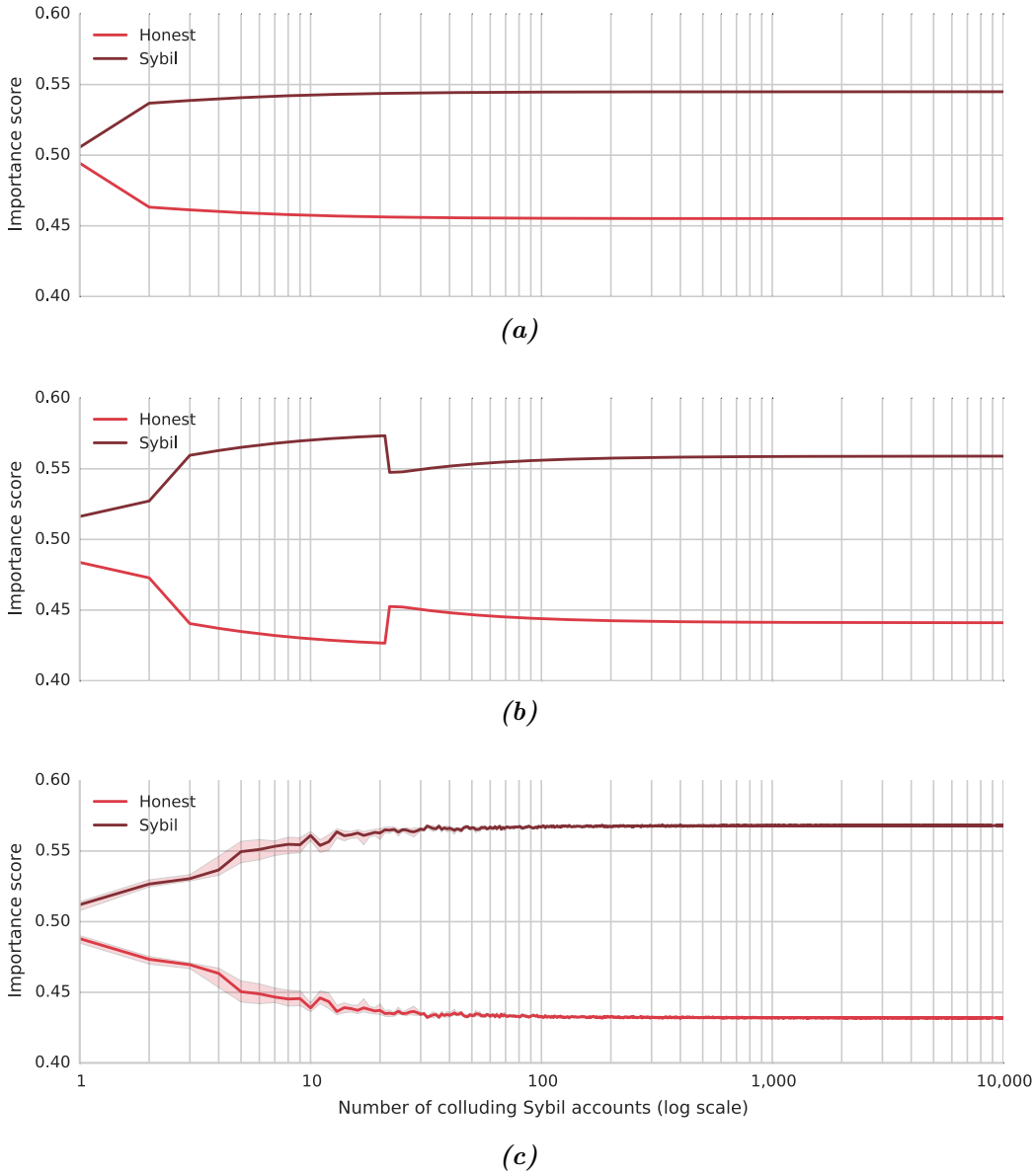


Figure 10: *Importances graphed for two actors with 800 million XEM with combined 40 million XEM sent via outlinking value transfers. The honest actor keeps his XEM in one account, whereas the Sybil actor controls multiple accounts and: (a) sends XEM around in a loop between controlled accounts, (b) vests the XEM in controlled accounts and then sends them all to a single master account, and (c) sends XEM from each controlled account to a random account. Importance scores are plotted on the y-axis and the number of colluding accounts for the Sybil actor is plotted in log scale on the x-axis. For (c), shading denotes 95% CI from 10 trials and solid lines denote the mean.*

7.7 Nothing-at-Stake Problem

Theoretically, algorithms for probabilistic Byzantine consensus that do not require the expenditure of external resources are subject to the *nothing-at-stake* problem[6]. The nothing-at-stake problem theoretically exists when the opportunity cost of creating a block is negligible. In other words, the cost of creating a block is so low that it is easy to create a block for all known forks of a block chain (including those forks that were self-created in secret). In contrast, the nothing-at-stake problem does not exist when Proof of Work is used for consensus because creating a block using such a consensus mechanism is resource-intensive.

In NEM, the cost of creating a block is negligible. To mitigate the nothing-at-stake problem, NEM caps the change in block difficulty and also limits the length of the chain that can be unwound during fork resolution.

As described in subsection 5.1: Block difficulty, the block difficulty change rate is capped at a maximum of 5%. If an attacker has considerably less than 50% of the harvesting power of the network, this will cause the block creation time of the attacker's secret chain to be much higher than that of the main chain in the beginning. This large time difference will make it unlikely for the attacker's chain to be better and accepted. In addition, the block chain unwind limit is 360 blocks (see subsection 5.4: Block chain synchronization), which prevents long-range forks from harvesters working on multiple chains.

7.8 Comparing importance to stake

As subsection 7.5: Calculating Importance Scores shows, the vested balance of an account is a large component of the importance score. Taking the vested balance as the "stake" used in currencies implementing the Proof-of-Stake (PoS) algorithm, it could be argued that PoS and PoI are similar. In order to explore the differences and similarities between PoS and PoI, the NEM and Bitcoin transaction graphs were analyzed. Bitcoin was chosen due to the relatively large size of its user base and transaction graph.

As of April 29, 2015, the NEM transaction graph had 1,456 harvesting-eligible accounts (see subsection 7.1: Eligibility for Entering the Importance Calculation). In October 2014, approximately one month of data from the Bitcoin network was downloaded and analyzed. 54,683 accounts were harvesting-eligible when BTC amounts were normalized to NEM. Amount normalization was done by multiplying the BTC amounts by the market cap ratio. Thus, 0.0177 BTC was the minimum amount of vested BTC for an account to be considered harvesting-eligible. Block heights were normalized to NEM by multiplying by

[6] See Vitalik Buterin's blog post on the subject for a good overview: https://blog.ethereum.org/2014/11/25/proof-stake-learned-love-weak-subjectivity/.

10.

Figure 11(a) shows importance scores and vested balances plotted on a log scale for each of the 1,456 harvesting-eligible accounts (sorted by vested balance, ascending) in the NEM transaction graph, and (b) for the 54,683 harvesting-eligible accounts in the Bitcoin transaction graph. As can be seen, while the vested balances are monotonically increasing, the importance scores are non-monotonic, demonstrating that accounts with lower vested balances are able to gain higher salience in PoI than in PoS in both transaction graphs.

Figure 12 shows the plotted NEM transaction network graphs for both normalized importance scores and vested balances in order to give an overview of the qualitative differences between PoI and PoS. Normalization was done to make the differences between scores with low and high values more visible. It was accomplished by scaling the importance scores and vested balances to sum to unity, taking the absolute value of the logarithm of that, mapping it to an exponential function, and then rescaling to sum to unity. As can be seen in the graph, vested balances give larger weight to fewer nodes, whereas importance scores have less noticeably larger nodes.

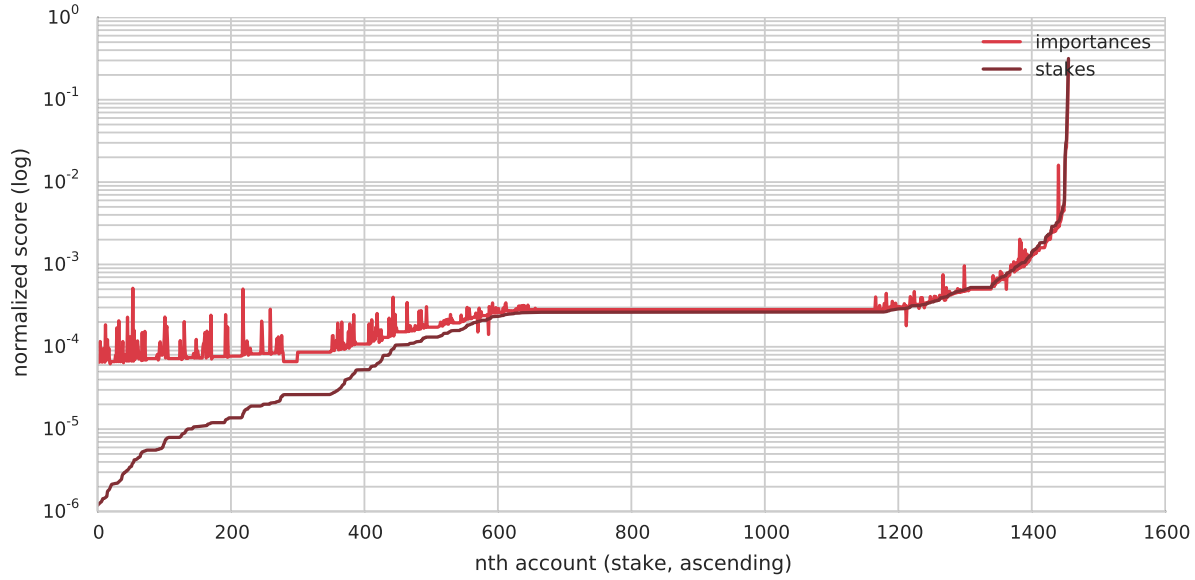
To quantify the differences in account saliences, all the accounts for NEM and Bitcoin were ranked (densely; accounts with the same score/balance were given the same rank) based on vested balances and importance scores and looked at the differences in ranks between the two metrics. Table 1: Differences between NEM account ranks for importance scores vs. vested balances. shows the results for accounts in the NEM transaction graph and Table 2: Differences between Bitcoin account ranks for importance scores vs. vested balances. shows the results of ranking Bitcoin accounts. Overall, NEM accounts ranked by importance scores were ranked approximately 338 ranks lower than when ranked by vested balances. The richer half of accounts decreased by an average of 443 ranks while the poorer half decreased by an average of 232 ranks. Bitcoin accounts were ranked an average of 67.5 ranks lower when ranked with importance scores than when ranked by vested balances. The poorer half of accounts gained an average of 2,814 ranks and the richer half lost an average of 2,950 ranks. This suggests that PoI gives less salience overall to richer accounts than PoS.

Table 1: *Differences between NEM account ranks for importance scores vs. vested balances.*

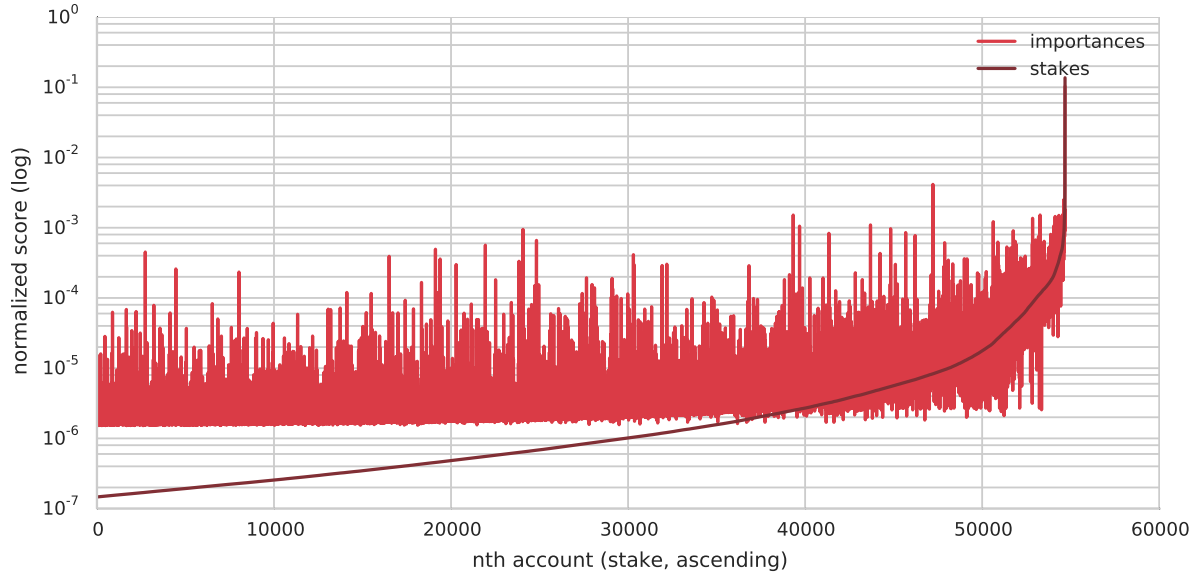
average rank increase for importance vs stakes (poor half):	-232.4
average rank increase for importance vs stakes (rich half):	-443.8
average rank increase for importance vs stakes:	-338.1

Table 2: *Differences between Bitcoin account ranks for importance scores vs. vested balances.*

average rank increase for importance vs stakes (poor half):	2814.6
average rank increase for importance vs stakes (rich half):	-2949.6
average rank increase for importance vs stakes:	-67.5

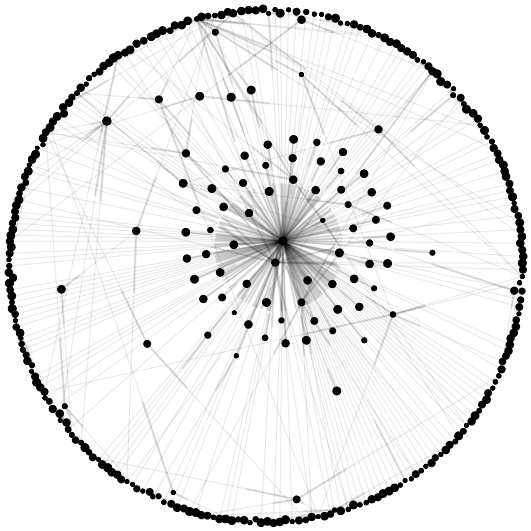


(a)

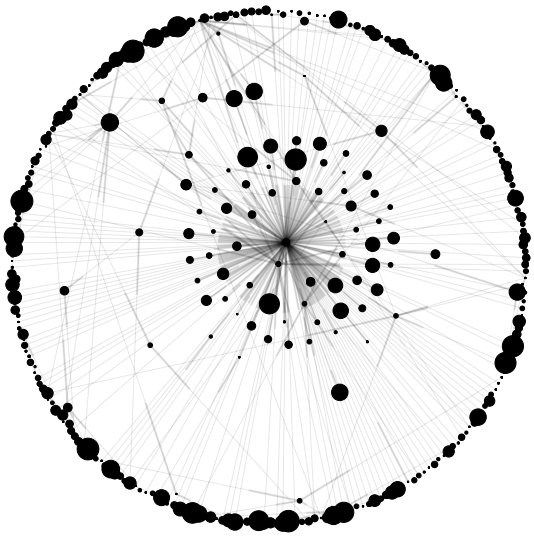


(b)

Figure 11: *Importance scores and vested balances for accounts in the harvesting-eligible subset of the (a) NEM and (b) Bitcoin transaction graphs are plotted. Importance scores and vested balances were normalized to sum to unity (1.0), with accounts sorted in ascending order. Graphs are plotted on the y-axis with a log scale, allowing a clear comparison between stake and importance scores, and accounts are on the x-axis.*



(a)



(b)

Figure 12: *NEM transaction graphs where node sizes denote normalized (a) importance scores and (b) vested balances. Normalization is described in subsection 7.8: Comparing importance to stake.*

8 Time synchronization

> You spend too much time on ephemeras. The majority of modern books are merely wavering reflections of the present. They disappear very quickly. You should read more old books. The classics. Goethe.
>
> - *Franz Kafka*

Like most other crypto currencies, NEM relies on time stamps for transactions and blocks. Ideally, all nodes in the network should be synchronized with respect to time. Even though most modern operating systems have time synchronization integrated, nodes can still have local clocks that deviate from real time by more than a minute. This causes those nodes to reject valid transactions or blocks, which makes it impossible for them to synchronize with the network.

It is therefore needed to have a synchronization mechanism to ensure all nodes agree on time. There are basically two ways to do this:

1. Use an existing protocol, such as NTP
2. Use a custom protocol

The advantage of using an existing protocol like NTP is that it is easy to implement and the network time will always be near real time. This has the disadvantage that the network relies on servers outside the network.

Using a custom protocol that only relies on the P2P network itself solves this problem, but there is a trade off. It is impossible to guarantee that the network time is always near real time. For an overview over different custom protocols see for example [12].

NEM uses a custom protocol in order to be completely independent from any outside entity.

8.1 Gathering samples

Each node in the network manages an integer number called *offset* which is set to 0 at start. The local system time in milliseconds incremented by the offset (which can be negative) is the *network time* (again in milliseconds) of the node.

After the start up of a node is completed, the node (hereafter called *local node*) selects up to 20 partner nodes for performing a time synchronization round. Only nodes that expose a minimum importance are considered as partners.

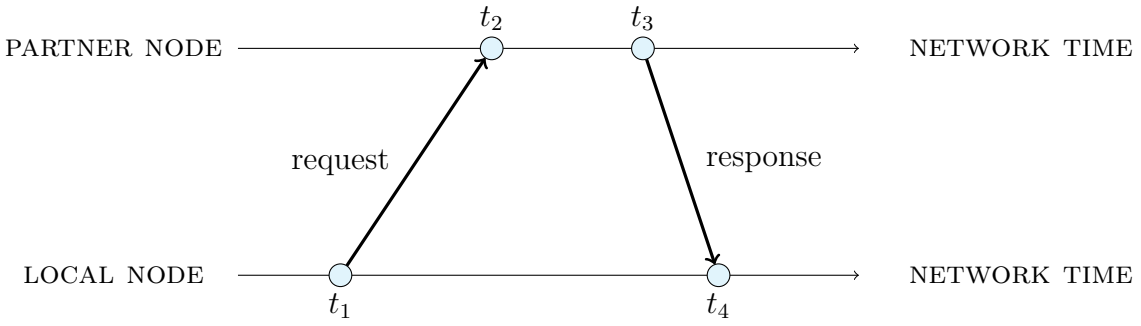


Figure 13: *Communication between local and partner node.*

For all selected partners the local node sends out a request asking the partner for its current network time. The local node remembers the network time stamps when the request was sent and when the response was received. Each partner node responds with a sample that contains the time stamp of the arrival of the request and the time stamp of the response. The partner node uses its own network time to create the time stamps. Figure 13 illustrates the communication between the nodes.

Using the time stamps, the local node can calculate the round trip time

$$rtt = (t_4 - t_1) - (t_3 - t_2)$$

and then estimate the offset o between the network time used by the two nodes as

$$o = t_2 - t_1 - \frac{rtt}{2}$$

This is repeated for every time synchronization partner until the local node has a list of offset estimations.

8.2 Applying filters to remove bad data

There could be bad samples due to various reasons:

- A malicious node can supply incorrect time stamps.
- An honest node can have a clock far from real time without knowing it and without having synchronized yet.
- The round trip time can be highly asymmetric due to internet problems or one of the nodes being very busy. This is known as channel asymmetry and cannot be avoided.

Filters are applied that try to remove the bad samples. The filtering is done in 3 steps:

1. If the response from a partner is not received within an expected time frame (i.e. if $t_4 - t_1 > 1000ms$) the sample is discarded.
2. If the calculated offset is not within certain bounds, the sample is discarded. The allowable bounds decrease as a node's uptime increases. When a node first joins the network, it tolerates a high offset in order to adjust to the already existing consensus of network time within the network. As time passes, the node gets less tolerant with respect to reported offsets. This ensures that malicious nodes reporting huge offsets are ignored after some time.
3. The remaining samples are ordered by their offset and then alpha trimmed on both ends. In other words, on both sides a certain portion of the samples is discarded.

8.3 Calculation of the effective offset

The reported offset is weighted with the importance of the boot account of the node reporting the offset. This is done to prevent Sybil attacks.

An attacker that tries to influence the calculated offset by running many nodes with low importances reporting offsets close to the tolerated bound will therefore not have a bigger influence than a single node having the same cumulative importance reporting the same offset. The influence of the attacker will be equal to the influence of the single node on a macro level.

Also, the numbers of samples that are available and the cumulative importance of all partner nodes should be incorporated. Each offset is therefore multiplied with a scaling factor.

Let I_j be the importance of the node reporting the j-th offset o_j and *viewSize* be the number of samples divided by the number of nodes that were eligible for the last PoI calculation.

Then the scaling factor used is

$$scale = \min\left(\frac{1}{\sum_j I_j}, \frac{1}{viewSize}\right)$$

This gives the formula for the effective offset o

$$o = scale \sum_j I_j \, o_j$$

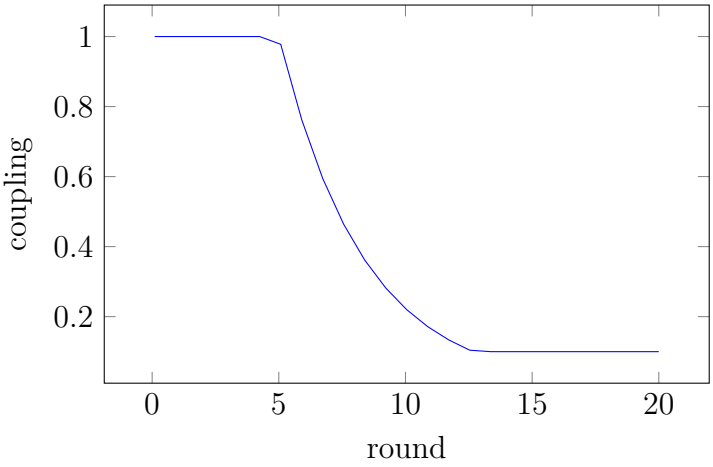


Figure 14: *Coupling factor*

Note that the influence of an account with large importance is artificially limited because the *viewSize* caps the scale. Such an account can raise its influence on a macro level by splitting its NEM into accounts that are not capped. But, doing so will likely decrease its influence on individual partners because the probability that all of its split accounts are chosen as time-sync partners for any single node is low.

8.4 Coupling and threshold

New nodes that just joined the network need to quickly adjust their offset to the already established network time. In contrast, old nodes should behave a lot more rigid in order to not get influenced by malicious nodes or newcomers too much.

In order to enable this, nodes only adjust a portion of the reported effective offset. Nodes multiply the effective offset with a coupling factor to build the final offset.

Each node keeps track of the number of time synchronization rounds it has performed. This is called the node age.

The formula for this coupling factor c is:

$$c = \max\left(e^{-0.3a},\ 0.1\right) \text{ where } a = \max(nodeAge - 5,\ 0)$$

This ensures that the coupling factor will be 1 for 5 rounds and then decay exponentially to 0.1.

Finally, a node only adds any calculated final offset to its internal offset if the absolute

value is above a given threshold (currently set to 75ms). This is effective in preventing slow drifts of the network time due to the communication between nodes having channel asymmetry.

9 Network

> For what is it? Nothing but a little blood and bones; a piece of network, wrought out of nerves, veins, and arteries twisted together.
>
> - *Marcus Aurelius*

The NEM network is comprised of NIS nodes. Each node is associated with a single primary account, which is used to authenticate responses by that node. This prevents an attacker from impersonating a node without having its private key even if he can spoof the node's IP address.

Each NIS node has the following configuration settings:

- `nis.nodeLimit` - the number of other nodes to which the local node should broadcast information
- `nis.timeSyncNodeLimit` - the number of other nodes the local node uses to synchronize its clock section 8: Time synchronization

Typically, `nis.timeSyncNodeLimit` should be larger than `nis.nodeLimit` in order to allow better time smoothing. This isn't too expensive because the data transferred as part of time synchronization is relatively small.

9.1 Node Protocol

NIS nodes communicate amongst themselves using a proprietary binary format by default. This format minimizes network bandwidth by compacting requests and processing by reducing the cost of serialization and deserialization. In fact, all NIS APIs support requests encoded in either the NEM proprietary binary format or JSON.

In order to prevent impersonation-based attacks, NIS nodes engage in a two part handshake when communicating:

1. The local node sends the request data and a random 64-byte payload to the remote node.
2. The remote node signs the random payload and sends the signature along with the requested response data back to the local node.
3. The local node checks the signature and only processes the response data if it can verify that the remote node signed the random payload.

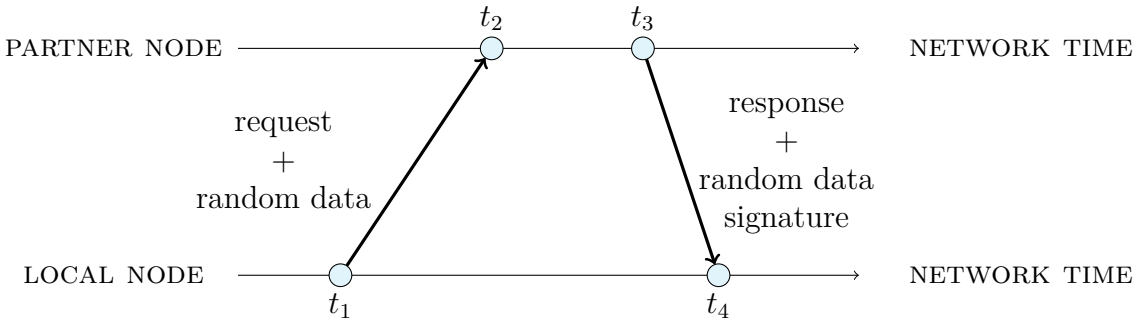


Figure 15: *Communication between local and partner node.*

9.2 Node Startup

When a NIS node is launched, the node processes the block chain and caches some data in memory to improve online performance. Once the processing is finished, the node is still not connected to the network because it is not yet booted.

A non-booted node is not associated with an account. This prevents it from being able to sign responses and prevents other nodes from being able to verify its identity.

In order to boot a node, the private key of a NEM account must be supplied to the node. This account is the primary account associated with the node. A delegated account can be used to boot a node in order to better protect the private key of the real account (see subsection 4.2: Importance transfer transactions).

9.3 Node Discovery

Once a node is booted, it connects to the NEM network and starts sharing information with other nodes. Initially, the node is only aware of the well-known nodes. These nodes are the same as the pre-trusted nodes described in subsection 6.2: Local trust value.

Over time, the node becomes aware of more nodes in the network. There are typically two ways this happens: via an announcement or a refresh.

9.3.1 Announcement

Periodically, a node announces itself to its current partner nodes and includes its local experience information in the request. If the node is unknown to the partner, the partner marks it as active and updates the node's experiences. If the node is known to the partner, the partner only updates the node's experiences but does not change its status.

9.3.2 Refresh

Periodically, a node asks its current partners to provide updated information about themselves and the state of the network.

First, the node requests update information about the remote node. If successful, the local node will update the remote's endpoint (necessary to support dynamic IPs) and metadata. This step fails if any of the following occur:

- The remote node returns a valid request from a different node
- The remote node is not compatible with the local node (e.g. the remote node is testnet but the local node is mainnet)

On success, the remote node is asked to provide a list of all of its current partner nodes. In order to prevent an evil node from providing incorrect information about good nodes and causing them to be blacklisted, the local node attempts to contact each reported partner node directly. It is important to note that a node's metadata will only be updated by metadata provided by that node itself.

As an optimization, blacklisted nodes are not refreshed. A node can be temporarily blacklisted if it returns a fatal error or the local node cannot connect to it. Periodically, blacklisted nodes will be removed from the blacklist and will be allowed to participate in the refresh operation again. If the original blacklisting was due to a non-transient error, the node will likely be blacklisted again.

9.4 Node Selection

Over time, as a result of the node discovery process, the local node will become aware of more nodes in the network. Eventually, the number of known nodes (including both well-known and other nodes) will be much greater than the number of partner nodes.

Periodically, a node recalculates its partner nodes. When this recalculation occurs, all known nodes are eligible to become a partner node, including nodes that have been detected as busy. With default settings, this recalculation will occur more frequently than the recalculation of trust values. By default, trust values are recalculated every 15 minutes.

The known nodes are weighted by their trust values and the partner nodes are randomly selected from among them. Nodes with larger trust values (which are more likely to be good nodes) are more likely to be chosen as partners. Nodes that have been minimally interacted with typically have trust values at or close to 0. In order to give these nodes

a chance to prove themselves and build trust, they are effectively given a small boost in trust so that they have the opportunity to be selected and participate in the network. 30% of trust is evenly distributed among nodes with less than 10 interactions.

In order to ensure that the network is connected, one adjustment to the random process is made. If the local node is a well-known node, it is additionally connected with all online well-known nodes. If the local node is not a well-known node, it is additionally connected with one random, online, well-known node. This ensures that all nodes are actively partnering with at least one well-known node.

References

[1] Lars Backstrom and Jure Leskovec. Supervised random walks: predicting and recommending links in social networks. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 635–644. ACM, 2011.

[2] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. In *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, pages 124–142, 2011.

[3] John R Douceur. The sybil attack. In *Peer-to-peer Systems*, pages 251–260. Springer, 2002.

[4] Nadav Eiron, Kevin S McCurley, and John A Tomlin. Ranking the web frontier. In *Proceedings of the 13th international conference on World Wide Web*, pages 309–318. ACM, 2004.

[5] Xinxin Fany, Ling Liu, Mingchu Li, and Zhiyuan Su. Eigentrust++: Attack resilient trust management. 2012.

[6] Tayfun Gürel, Luc De Raedt, and Stefan Rotter. Ranking neurons for mining structure-activity relations in biological neural networks: Neuronrank. *Neurocomputing*, 70(10):1897–1901, 2007.

[7] Bin Jiang. Ranking spaces for predicting human movement in an urban environment. *International Journal of Geographical Information Science*, 23(7):823–837, 2009.

[8] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. *Proceedings of the 12th international conference on World Wide Web*, pages 640 – 651, 2003.

[9] A.N. Langville and C.D. Meyer. *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Princeton University Press, Princeton, USA, 2006.

[10] Athanasios N Nikolakopoulos and John D Garofalakis. Ncdawarerank: a novel ranking method that exploits the decomposable structure of the web. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 143–152. ACM, 2013.

[11] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: bringing order to the web. Technical Report 1999-66, Stanford InfoLab, Stanford, USA, November 1999.

[12] Sirio Scipioni. *Algorithms and Services for Peer-to-Peer Internal Clock Synchronization*. PhD thesis, Universit'a degli Studi di Roma „La Sapienza", 2009.

[13] H Shiokawa, Y Fujiwara, and M Onizuka. Fast structural similarity graph clustering. In *The 6th Forum on Data Engineering and Information Management (DEIM2014)*, 2014.

[14] Xiaowei Xu, Nurcan Yuruk, Zhidan Feng, and Thomas AJ Schweiger. Scan: a structural clustering algorithm for networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 824–833. ACM, 2007.