Sync

The most basic building block of any blockchain is a ledger to keep track of state transitions. If all we care about is keeping this ledger on our local machine, then there's not much more we have to do, but things tend to be more interesting whenever there are more parties involved. As soon as there are multiple parties involved, a distributed ledger is needed, which requires us to communicate with others and exchange information about state transitions in order to synchronise their local state machines.

This document describes how a set of peers can exchange information about state transitions in a settings where peers might not trust each other or could possibly even be acting maliciously.

With that in mind, we're going to have to come up with a protocol that minimises attack surface while not (significantly) hampering the speed with which information is disseminated in the network.

There are a couple of dimensions along which we can tune our protocol:

- network bandwidth usage
- network latency for transaction/block propagation
- reliability (in the presence of adversaries)
- memory/CPU usage

but as is almost always the case, we cannot have the best of all of them.

Ideally, our protocol has knobs which let users configure it to fit their needs, e.g. a node running on a mobile phone might want to reduce bandwidth at the cost of increased latency or on the other side of the spectrum a strong node might elect to reduce latency by using more network bandwidth and increased CPU usage.

Transport protocol

This initial version is going to have nodes using TCP connections. All communication will be encrypted and authenticated to ensure both confidentiality and integrity, which prevents adversaries from interfering with the protocol and de-anonymisation of nodes to a degree.

For encryption and authentication we are going to use the Noise protocol, with the exact protocol name being `Noise_XK_25519_ChaChaPoly_Blake2b`, i.e. `XK` for handshakes, DH over `Curve25519`, `ChaChaPoly` for symmetric encryption and the `Blake2b` hash function. `XK` as a handshake pattern means that the initiator of the handshake sends their static key to the responder and the initiator knows the static keyof the responder.

Each node has a static `Curve25519` key pair for P2P communication. The peer discovery is bootstrapped by having a set of Aeternity peer addresses in the node configuration and the node tries to connect to these peers after starting up the node. Peer addresses looks like: `aenode://pp_ttZZwTS2nzxg7pnEeFMWeRCfdvdxeRu6SgVyeALZX3LbdeiWS@31.13.249.0:3015`.

(***TODO***: Spell out the full protocol including key schedule etc.)

Peer to peer network

(***TODO***: this should probably be the overarching topic for this document and sync being a sub topic)

Given that all nodes participating in the network are going to hold all available data, there is no need to have a structured overlay network.

An overview of the P2P-message protocol is here.

Bootstrapping

Before any node can start the process of downloading the blockchain, it needs to discover suitable neighbours, which are part of the peer to peer network. We consider two different ways to get this process bootstrapped:

1. have a central service, that hands out an initial set of neighbours, e.g. via a DNS lookup
2. ship the node software with a set of initial neighbours

Both options have advantages and disadvantages. Number one means that the set of initial neighbours can be easily rotated without having to update any configuration files but it depends on fairly centralised infrastructure. Option two does not rely on DNS but requires updates to the node software packages in order if the initial neighbour set needs to change.

We will choose one of the following strategies to acquire our initial entrypoints for the network, in order of preference:

1. consult local database of peers the node has talked to before and try to connect to them
2. connect to a list of known peers with DNS entries
3. rely on hard-coded nodes that come with the node package

Neighbour selection

After a node has been bootstrapped, it should try to keep a fairly stable neighbourhood and thus try to persist peers across restarts.

To describe the selection process, we need to come up with strategies for neighbour replacement and a preference function used to pick a possible new peer out of a set.

Choosing a selection strategy has a big impact on the overall health of the network, since it influences the node degree, which in turn controls resilience and message overhead.

There's rich literature analysing different strategies based on values intrinsic to the peer to peer overlay network, e.g. peer age or behaviour metrics, or on values of the underlying network, e.g. RTT or geographical location.

Replacement strategies based on age can be put into two different categories:

- *active*:
- *passive*:

A number of different selection strategies have been discussed in the literature but in order to choose the right one we need to first come up with requirements.

Routing

In the Bitcoin and Ethereum networks, routing is of no concern, given that it is generally impossible to say, who will produce the next block and as such a user will just post a transaction to the network in the hopes of reaching all miners, who can then pick it up.

Given our usage of the Bitcoin-NG[7] protocol, a routing layer might be of interest for users, who have tight timing requirements, or just for the improved user experience by a potential decrease in latency when a transaction can be routed directly to the current round leader versus flooding the network.

The addition of routing will be addressed in future versions of this document.

Churn

There are many studies observing and studying churn patterns in peer to peer networks such as Gnutella or BitTorrent. While these are certainly valuable to consider in this context, we can assume that the usage patterns differ quite a lot, given that they are mostly used for sharing relatively small, self- contained files.

Donet et al [8] did an early study of the bitcoin peer to peer network in 2013, finding "872648 different IP addresses corresponding to machines running Bitcoin nodes." but of which only a staggering 0.66% (5759) were still reachable after their observation period of 37 days. bitnodes has been trying to keep track of the peer to peer network size, starting in May 2016 with around 6500 nodes.

Connection

incoming/outgoing

whitelist/blacklist

Incentives

The incentives for nodes to share transactions and blocks with other nodes are weak and there is a body of research suggesting that for Nakamoto style consensus it might even be in a miners' interest to not share new blocks right away. [1][2] These shortcomings also incentivise centralisation of mining power, which is something we would like to avoid.

Another example could a mining pool, which wants to receive information about new blocks and transactions as fast as possible and thus might forego thoroughly checking blocks for validity to reduce latency. This is clearly not in the interest of the network.

(***TODO***: Come up with something similar to/based on TorCoin, or at least mention it and give outlook for how it might be integrated.)

Block/Transaction Synchronization

At startup (after discovering at least one other Peer) the node will synchronize the chain, and the current list of unconfirmed transactions.

Synchronizing the chain is (potentially) a big task, and the node will utilize all its connected peers to fetch blocks and transactions.

(***TODO***: Write a detailed description of the Aeternity node implementation.)

Synchronizing the unconfirmed transactions is a smaller task and the node will pick one other peer and synchronize (only the missing) transactions from this peer. The synchronization uses MP-trees - a more detailed description is here.

Block/Transaction Propagation

- with bitcoin-ng always sending full blocks would be a giant waste
- announce new tx/block to peers
- peer asks for data if they don't have it
- don't have to send full block with all tx
- send header first
- consider overhead for sending out single tx vs. batching

Proof of work puzzle

Node operators have the option to require connecting peers to solve a proof of work puzzle as a means of rate limiting. They should make use of this option if they believe to be the target of an eclipse attack or a denial of service.

The solution to such a puzzle could either be generated offline or online both of which have their merits.

Being able to generate the solution offline would give resource constrained nodes the possibility to easily participate but would also allow an attacker to generate many solutions before an attack. An example for such an offline scheme might be to find a solution to `Blake2b(CuckooCycle(Pubkey || Nonce)) > DifficultyTarget`. The peer would, upon connecting, present the nonce and the receiver of the connection could then verify that the static key used for the noise handshake concatenated with the nonce is indeed a valid solution. (***TODO***: this might require a consensus value otherwise nodes wouldn't know what to expect)

Juels and Brainard [6] were the first to introduce an interactive scheme requiring the solution of a computational puzzle to combat denial of service attacks.

The first message a connecting node needs to sent after establishing the secure session is send its proof of work puzzle solution.

- this could either be interactive
    1. initiator connects to node

2. responder sends challenge
3. initiator disconnects
4. initiator computes solution
5. initiator connects again and provides solution

Problems could arise if an overly difficult pow puzzle is required by most nodes because it might prevent the majority of people from running nodes and hurt the health of the network.

Reputation

The sync protocol uses a very simple reputation system for peers, which will only be used to punish peers sending garbage or misbehaving. Reputation is local to a node and not shared with anyone. Once an offending node reaches some threshold, e.g. 0, the offended node MAY terminate the connection and SHOULD reject any further connection attempts by the offender for a while, e.g. 24 hours, if they choose to punish them.

Garbage in this content would be:

- invalid blocks
- invalid transactions
- ...

If the offender is motivated and willing to bring up more nodes, then the node under attack MAY opt for connecting nodes be required to solve a proof-of-work puzzle reduce before they are accepted.

Configurables

- pow target
- pow wait timeout
- (listener interface, port)
- connection timeout
- ## overall connections
- ## incoming connections
- gossip pool size
- peers -> (blacklist, whitelist)

Threat Model

Any threat model comes with a set of assumptions and attack vectors that are out of scope.

The first assumption is, that our distributed ledger is a blockchain using Nakamoto style consensus with proof of work for leader election, with any malicious miner not possessing more than 25% of the mining power of the network. This guarantees the basic integrity of our ledger. (***TODO***: Properly address selfish mining et al)

The protocol described here is not used to transmit any secrets, such as cryptographic keys controlling coins, but does make use of cryptographic keys itself. For an adversary to get ahold of these means that they would be able to initiate the protocol with a node using them. Given that the protocol is only used to transmit public information and generating new keys is easy, this should only be a minor inconvenience. (***TODO***: This changes if we use them for authentication as well)

We do not consider any physical threats to nodes participating in the protocol, under the above assumption and because it is generally considered to be a very hard problem, trying to design protocols, which can deal with compromised endpoints[4]. Usage of secure enclaves and other special purpose hardware can be considered for mitigation but will not be discussed here.

That is, we operate under the "Internet Threat Model" as outlined in Section 3 of RFC3552[4]:

> In general, we assume that the end-systems engaging in a protocol exchange have not themselves been compromised. Protecting against an attack when one of the end-systems has been compromised is extraordinarily difficult. It is, however, possible to design protocols which minimize the extent of the damage done under these circumstances.

> By contrast, we assume that the attacker has nearly complete control of the communications channel over which the end-systems communicate. This means that the attacker can read any PDU (Protocol Data Unit) on the network and undetectably remove, change, or inject forged packets onto the wire. This includes being able to generate packets that appear to be from a trusted machine. Thus, even if the end-system with which you wish to communicate is itself secure, the Internet environment provides no assurance that packets which claim to be from that system in fact are.

We restrict this model even further and exclude TCP and IP level (distributed) denial of service attacks, e.g. SYN floods or any sort of amplification attacks.

Transport

Transport messages are encrypted and authenticated using the Noise protocol. For an in-depth discussion of the Noise protocol and its security guarantees, please refer to the documentation, specifically sections 7 and 14. Using encryption and authentication covers man-in-the-middle attacks, eavesdropping,

A Denial of Service (DoS) attack can be mounted at different layers, e.g. at the protocol or the node level.

- slowing down block propagation
    - incr. risk of forks
    - makes selfish mining even more profitable
- resource exhaustion for single node by using up all connections/filling up RAM

Security:

- eclipse attack -> accepting double spends

Privacy:

- anyone in the same network can basically see everything you do without encryption

Network level Distributed Denial of Service (DDoS) attacks, that try to take nodes offline by saturating their connection to the internet, are out of scope, although it could be argued that using mix networks hide the IP addresses of parties communicating and thus might mitigate one DDoS angle.

Resource exhaustion

Eclipse Attacks

Whenever an adversary manages to surround an honest node with adversarial nodes then we speak of an eclipse attack.

In order to prevent malicious parties from flooding the network with requests and potentially eclipse new nodes, we will require nodes to attach the solution to a proof of work puzzle whenever they try to initiate a connection to another node. This solution

Privacy

References

[1]: Eyal, Ittay, and Emin Gün Sirer. "Majority is not enough: Bitcoin mining is vulnerable." International conference on financial cryptography and data security. Springer, Berlin, Heidelberg, 2014.

[2]: Nayak, Kartik, et al. "Stubborn mining: Generalizing selfish mining and combining with an eclipse attack." Security and Privacy (EuroS&P), 2016 IEEE European Symposium on. IEEE, 2016.

[3]: Ghosh, Mainak, et al. A TorPath to TorCoin: proof-of-bandwidth altcoins for compensating relays. NAVAL RESEARCH LAB WASHINGTON DC, 2014.

[4]: Rescorla, E., and B. Korver. "RFC 3552: Guidelines for writing RFC text on security considerations." Internet Society Req. for Comm (2003).

[5]:

[6]: Juels, Ari, and John G. Brainard. "Client puzzles: A Cryptographic countermeasure against connection depletion attacks." NDSS. Vol. 99. 1999.

[7]: Eyal, Ittay, et al. "Bitcoin-NG: A Scalable Blockchain Protocol." NSDI. 2016.

[8]: Donet, Joan Antoni Donet, Cristina Pérez-Sola, and Jordi Herrera-Joancomartí. "The bitcoin P2P network." International Conference on Financial Cryptography and Data Security. Springer, Berlin, Heidelberg, 2014.