HOCHSCHULE MITTWEIDA
UNIVERSITY OF APPLIED SCIENCES

MASTERTHESIS

Mr
Jonas Bentke

Incentivisation structure for the Incentivised Remote Node Network protocol

2020

Faculty of **Applied Computer and Life Sciences**

MASTERTHESIS

Incentivisation structure for the Incentivised Remote Node Network protocol

Author:
Jonas Bentke

Study Programme:
Distributed Ledger Technologies

Seminar Group:
BC18w1-M

First Referee:
Pr. Dr. Andreas Ittner

Second Referee:
M. Sc. Steffen Kux

Mittweida, 10 2020

Bibliographic Information
Bentke, Jonas: Incentivisation structure for the Incentivised Remote Node Network protocol, 40 pages, 9 figures, Hochschule Mittweida, University of Applied Sciences, Faculty of Applied Computer and Life Sciences
Masterthesis, 2020

Abstract
This work is an analysis of the IN3 protocol and suggests solutions to the free rider problem, server scoring and selection as well as payment incentives.

I. Contents

II. List of Figures

1 Introduction

Since Ethereum's inception, the authors were always concerned about network capacity, because of this they envisioned a light client that is able to validate blockchain data without the need of downloading a lot of the chain. To this day the Light Ethereum Subprotocol (LES) [24] is still under development. Blockchains Development Labs (formerly slock.it) at Blockchains LLC is currently developing a similar protocol that removes some of the Obstacles present in the LES and is called the Incentivised Remote Node Netwoek (IN3). This work is taking a closer look at some of the obstacles that can arise when pushing IN3 into the production stage.

While LES is running inside the p2p system of Ethereum, IN3 is taking a different approach and is building a second layer on top of it. The layers are connected by the IN3 servers that connect directly to an Ethereum node. IN3 clients are connecting to those servers to make requests pertaining to blockchain information. In order for this protocol to work, there must be some incentive for the servers to process those requests. Introducing payments into a protocol can be challenging and complex. The protocol must now not only be resistant to software attacks but also to loopholes in the economic model. Free riders and monopolies can dominate and even destroy small markets. How the client is picking a server plays a big part in enabling healthy competition among the providers. How and when the client will pay also depends on trust build among the participants and whom the client chose to interact with.

This document starts by providing a summary of specific Ethereum features that are needed to understand how IN3 works. Then it contains a short explanation of IN3 and why it is needed. The economics of request distribution and a solution to the free rider problem is covered in Chapter 4. Chapter 5 covers server scoring and selection. It highlights the specific attributes that need to be considered for an adequate scoring model. Optimal selection of servers is briefly considered, followed by a validity estimation for a consensus based selection. Finally, this document will cover the economics of payments within the in3 protocol and how reputation systems can be used to prevent monopolies and helps to enable post pay systems. This will be used to highlight a selection of layer 2 solutions to implement payments under specific constraints.

2 Ethereum basics

This chapter highlights the most important features of the Ethereum Blockchain, it includes the key features of IN3, and provides an overview of the topics covered in this thesis.

Ethereum is a distributed general purpose state transition machine. The state is made up of key value pairs where the keys are addresses and the values are account states. Every account state contains key information such as the current program counter (nonce), its balance, storage root, and when the state points to a smart contract it is also the *codeHash*. A transaction can change individual or multiple account states, which then changes the state of the overall machine. The Ethereum virtual machine is controlled by a set of commands which define the rules of changing the state. How this is done is not important for the IN3 use case, suffice to say that every participant runs new transactions locally on their own virtual machine to verify the validity of the transaction and the resulting state.

In order to reach a consensus on the order of executed transactions, block generators gather and summarise transactions into blocks. Additonally the header of the block contains several important hash values. They contain the hash of the previous block header and the *mixHash* which are both needed for block verification, the *stateRoot* which is the root hash of the state tree, the *transactionRoot* which is the root hash of the transactions included in this block, and the *receiptsRoot* which is the root hash of the transaction receipt tree.

In order to understand the importance of the previously mentioned root hashes, an explanation of a merkle tree in required. A merkle tree is a data structure that stores key value pairs and works similar to a normal radix tree (trie). It has a root node, normal nodes or branches, and leaves. Ethereum in fact uses Merkle Patricia trees, which optimise the structure and those optimisations are not inline with the scope of this thesis as knowledge of normal merkle trees is sufficient. Figure 2.1 shows how a merkle tree is structured. Each node is in essence a list of key value pairs, where the value is a hash of another node. With that hash, we can look up another node in the database which contains itself a list of key value pairs. The size of the list is dependent on the alphabet used, in this case, 16 as its hexadecimal. The depth of a tree is dependent on the key size which is here 32 bytes.

The example of figure 2.1 shows that more clearly. Here the value of the key *0x12F* is the one requested. To do that the tree needs to be traversed by getting the root node and check index 1. Index 1 contains the hash value of the child node linked to that position. The value can be found by looking up the hash in the database. That position contains yet another list with 16 hash values. Here the search key now specifies the next value to be at index 2. At the end it can be found that index F contains the address to reach the value stored there and which shows that it is a leave. The content of the leave is then the value for the key *0x12F*. Hashes are generated from the contents of the node. The example shows that Hash 3-1 is generated directly from the data. If the

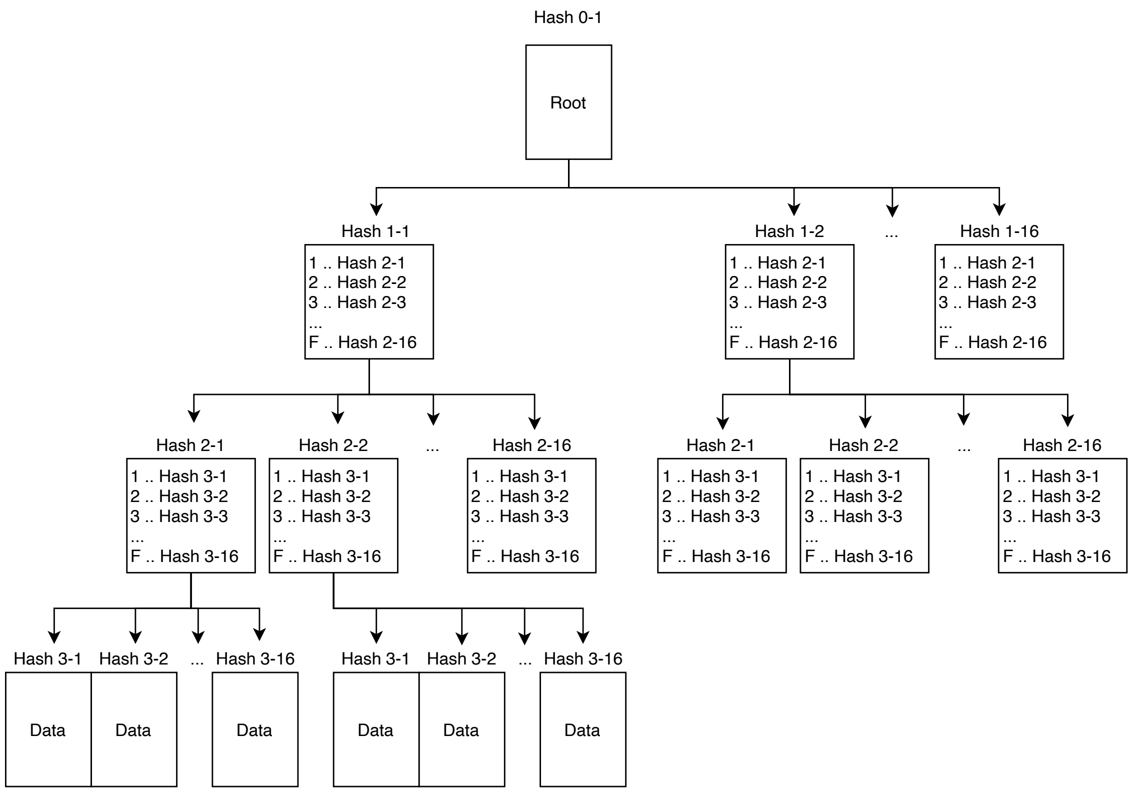

Figure 2.1: General depiction of a Merkle Tree

data changes, the hash changes and is propagated upwards to the correct key index. This will then change Hash 2-1 and so on until the root hash has changed.

The reason for having such a trie is to enable quick and cheap data verification. Data can be verified by delivering the hashes that lead to the root hash. For example on the transaction trie, an Ethereum node can easily prove that a transaction is in the block, by delivering that transaction and all hashes to the requester. This technology is used by light clients that do not store the complete blockchain, but instead only the list of block headers. If the user needs to know when a transaction has been executed, and at which specific block, then a request for a merkle proof can be made from a node. That merkle proof can then be verified against the *receiptsRoot* contained in their copy of the blockheader. Blockheader verification is easy as long as they have the complete list stored locally.

3 IN3

3.1 Why IN3 is needed

Running an Ethereum client comes with its own challenges. At the time of writing a full archive node consumes 4 TB of storage space and a normal full node is still around 300GB with pruning[1] active. In order to synchronise with the network the client will need to be run on an SSD. These requirements represent a huge barrier for anyone that wants to connect to the Blockchain.

To enable more people to access the network, client providers have come up with a couple of different technologies to facilitate smaller devices like light clients. But, even if those clients would be enough for stationary devices most users and with that most use cases require the client to run on a mobile device. Going a step further and covering the use cases of the IoT industry the client would actually be required to run on a much smaller device. Light clients can not efficiently run on these devices for multiple reasons. One reason is the problem of synchronicity. In order to use the client, it must get up to date with the latest blocks which can take some time depending on the last update. This would simply not be accepted by any user for any app, especially for payments. Another reason is bandwidth. Keeping the blockchain up to date requires a lot of data to be exchanged. That can be very bothersome for mobile users that do not run on a data flat rate.

The current standard to circumvent all of these problems is to run a remote client. This is where user software does not run its own blockchain client anymore but simply connects to a server that does. This way all the initial problems such as bandwidth and waiting times are solved. It however, comes with a caveat. Blockchain technologies' big advantage over traditional cloud-based infrastructure is decentralisation. Decentralisation not only in terms of connectivity but also in terms of Blockchain governance. The first issue is very clear. If a company must run its own backend to connect to the blockchain it might as well run its own database to begin with. It is much cheaper than storing the data on a blockchain. Thus many small companies decided to connect to an external service that handles the blockchain connections for them.

The market leader in this field is Infura. As described in [21] forking is a new governance tool introduced through blockchain. It enables nodes to split from the original network if they disagree with an update. It is a vital and important part of the ecosystem since update creation is already centralised and client diversity is very low. Through remote nodes, however, even forking becomes centralised. Take a provider like Infura. According to recent surveys, 63% stated that they use Infura for their dApp. Those 63% also represent the revenue made by miners. Infura would have huge bargaining power with the miners on whether to accept or decline the proposed fork. Miners rely on the traffic Infura brings to the network and Infura relies on the miners for process-

[1] Pruning is a technique used by ethereum clients to delete unneeded data. That data could be older states or address storage that has only been touched but not used

ing power. Mining is, however, a competitive business and thus Infura would just need to collude with a fraction of the total mining power.

To cover the problems of usability and decentralisation, Blockchains developed a client that solves those issues.

3.2 How IN3 works

To achieve the attributes of usability and decentralisation, IN3 is building a second layer on top of existing blockchains (we will further use Ethereum as the subject of explanation). This second layer consists of servers that access blockchain nodes directly. Each server must be registered in a special registry smart contract. In order to register, the server operator must leave a security deposit that can be slashed to punish any misbehaviour. Clients have to get a node list either from the set of boot nodes or must be preconfigured with one. They can then start to send requests to servers on that list.

Since regular answers can not be validated by the client itself, the server has to send some form of proof along with its response. IN3 is using the merkle trees and corresponding proof data to verify the validity of the response, however not the validity of the underlying blockhash. We already mentioned that traditional light clients need to verify the complete list of blockheaders and the corresponding blockhashes. Some requests such as the aforementioned blockhashes can not be checked through merkle proofs. Other examples for these kinds of requests are those that rely on the context of the host machine. An example would be the latest Block Number, since the server might not have received the latest block number yet or is on a micro fork. These requests are vital to the network and so the server must serve them

All the client can do at that point is to query those requests from multiple sources and compare them with each other. This involves a certain risk of being scammed into believing false information, and a more extensive risk analysis is discussed in chapter 5.1. Avoiding malicious Server nodes is ensured through an internal ranking system that downvotes servers that send responses that do not match with the other query results. Server nodes will be ranked by each client internally. This ranking includes different attributes like response time or correctness. This form of ranking is necessary to improve performance from the client's perspective since it's only choosing the best in spot server node for its requests. Slower or untruthful nodes will lose traffic and thus money and must either improve their score or drop out of the system. Chapter 5 will cover how the ranking system is built up.

Multiple problems arise if those features are not implemented correctly. Some of these problems may exist only in theory now. However, because the problems are present on the protocol level they can rapidly become a reality. The problems presented manifest in two broader categories, which are monetary problems server side and monetary problems client side. Each server has costs and those costs must be covered. If they are not covered (not even necessarily through monetary means) the incentive is lost to run a server and the system collapses. On the other hand, clients also have costs that arise not only from potentially paying servers but also in the form of time conflicts, for example if a client has to ask hundreds of servers before getting a reliable answer.

The following sections will describe those problems in more detail and offer solutions that can help with the implementation later on. The next chapter will cover unvalidated requests and how IN3 has a problem with free riders. Following is the client scoring and how it should be built up as well as how servers should then be selected based on that score. Lastly, we will look at the payments of request, how they should be made, and what pitfalls to look for. That section also includes different payment technologies and how they are currently implemented.

4 Unvalidated Requests

4.1 Tragedy of the commons

The tragedy of the commons is termed by the American ecologist Garrett Hardin in his equally named paper that he based on a pamphlet by William Forster Lloyd [3]. In his paper, he describes a setting where a common good accessible to anyone is overused by some, but the resulting damage is carried by all. One of many examples is overfishing. Each rational acting fisher wants to maximise his profits and thus fishes as much as he can. For every other fisher that reasoning is sounding as well and thus overfishing occurs where species go extinct and the seas will become empty in the long run. This problem can also be seen in very localised events. During a marketing campaign of a local gas station, every citizen was entitled to 20 liters of free gas [5]. This resulted in multiple km long traffic jams for all citizens, even those that did not participate. Another example that is brought forward is the reverse of the above. Hardin wrote that

> [...] the tragedy of the commons reappears in problems of pollution. Here it is not a question of taking something out of the commons, but of putting something in-sewage, or chemical, radioactive, and heat wastes into water; noxious and dangerous fumes into the air; and distracting and unpleasant advertising signs into the line of sight. The calculations of utility are much the same as before. The rational man finds that his share of the cost of the wastes he discharges into the commons is less than the cost of purifying his wastes before releasing them. Since this is true for everyone, we are locked into a system of "fouling our own nest," [...]

Certain IN3 features enable the same problem for the protocol. Requests are separated into two categories as previously described. The problem lies in the dependability of those requests. Clients that want to send a transaction within the ethereum network for example, have to request non-provable values from the node such as the current nonce or balance. But since those values are depending on the server's personal scope of the blockchain it is not possible to hold the server accountable for any false information. Servers currently have the problem that clients don't have a unique identification that couples them to the real world. Clients can change their identity very easily and thus appear to be a new user rather than one already known. If a client requests the current blockhash for example, the server can not know if further requests will be made that he can charge for. Rational Clients could then use only a subset of servers for free unvalidated requests and use that information to do validated requests on the other servers, thus avoiding additional costs that the server that serves unvalidated requests would have put on top of the paid validated requests.

Even if the server would want to be paid for unvalidated requests, from a client's perspective payments are not feasible for those since the client can not prove data validity. Rational servers would then instead of finding the true answer to the request send false information which is easier to procure since clients can not know that it is in fact not the correct answer. It is shown in another chapter how that problem can be solved, however, one unique attribute of unvalidated requests is that they depend on server context and thus, must not intentionally be a lie. Servers that have the latest blockhash simply cashed for performance can not be differentiated between servers that would request the blockhash every time but run on a micro fork or simply lack behind in blocks. Thus even if the client would know that it is a wrong response it can not punish the server for its doings and therefore, the original problem still exists.

This problem thus represents a form of the tragedy of the commons or otherwise known as the free rider problem that is similar to the pollution example. All clients can use up the resources of the servers. The costs are however only carried by a few.

4.2 Solutions

Solutions to the tragedy of the commons are generally to either have a regulatory body in the form of a government that controls access rights and usage permits or to privatise the commons. To find an appropriate solution for the IN3 case we need to first define the actors more closely. The commons as discussed here refer to the capacity of all IN3 server to serve requests. It might seem at first glance that in fact, this capacity is not a common at all since each server can individually decide whether to provide requests or not. In fact, later in this section a discussion of a case where a server will only accept validated requests and avoid the cost of serving unvalidated ones is raised. However, two attributes of the current implementation strongly suggest the request pool to be a common.

As previously mentioned, the execution of validated requests depends on the information received from unvalidated requests. Servers can thereby not make any money without there being a free provider for that information. Secondly, Servers can not effectively distinguish between clients, because clients are not bound (and in fact are discouraged) to use the same server for dependable requests. In this setting, servers are the only available information provider and are required therefore to provide free requests. This description fits with Elinor Ostroms definition of common-pool resources (CPR) that separate the actual resource from property rights [6].

CPR's come with two attributes: "(i) exclusion of beneficiaries through physical and institutional means is especially costly, and (ii) exploitation by one user reduces resource availability for others".

(i) is fulfilled through the problem of identifying individual clients and (ii) is self-evident, that when one client uses a request, the overall pool of available request slots decreases. Servers are thus acting as property owners that are forced to provide access to its capacities and those capacities are then used as a common. Clients are then the consumer of the common. One last actor can be defined as the Government. In IN3's case that would be the protocol designer/software developer. The developer can regulate certain aspects of consumption by pushing code updates or by defining protocol rules. Since he is not however the "property" owner, it can not be safely assumed that those rules and protocols are actually enforced.

4.2.1 Non governmental approach

It is now discussed what happens if a self governing approach or a non-governmental solution would be pursued. This would mean that clients find a way to self organise usage rights over server capacities. In order to do so, several criteria need to be checked according to Ostroms. These include resource boundaries, the threat of resource depletion, a thick social network with established norms, and lastly a way to incentivise or punish behavior.

Resource boundaries are very clear, as it is simply the sum of the overall network capacity. It is not known how big that capacity is in general but it can be estimated by looking at the onchain node list. Threat of depletion will at the latest show when it is occurring but can also be indicated in advance through average response times. Servers that are already at capacity will reject further responses and the client will need to find another server. For now, we will assume that there is a sufficient server scoring run by clients to discover resource depletion in the early state.

Social networks however are clearly not a given. This is partly because the user group is so diverse. IN3 is going to be used by end customers running it on their smart phone as well as big industrial companies running it on millions of smart sensors. This makes it inherently difficult to establish a baseline of norms and rules. Additionally, the fact that everyone is anonymous within the system makes it nearly impossible to properly punish over usage. Just getting to know the exact amount of consumers is impossible from the client side. It is clear now that letting the clients self organise is simply not possible since they would just over consume due to a lack of rule enforcement.

4.2.2 Privatisation

The reason why unvalidated request had to be classify as a common was that the exclusion of beneficiaries was very difficult. This hurdle must be taken in order to privatise the request and thus incentivise the servers themselves to make them sustainable. It will now described how a game under the current assumptions and analyse each participant's strategy to show more clearly what we mean. A game consist of the following ingredients:

- A set of players $P = \{i, j\}$
- A set of possible strategies S where S_i represent the set of strategies for player i and s_i a particular strategy for player i and s is a particular strategic profile of a play of a game
- A payoff function denoted as $u(s)$ where $u_i(s_i, s_j)$ is the payoff for player i when i chooses the strategy s_i and player j chooses the strategy s_j

In this case only one shot games are considered. Reasons for that are simply the fact that a client does not have to send multiple requests to the same server. Even a limited amount of Servers would not change that fact since the client can change his identity after each game with no extra cost. As mentioned before, clients are encouraged to send dependent requests to different servers to avoid an attack vector where the server can potentially use the gathered information for front running for example. Strategies are defined as follows:

- $s_{\text{client}} = \{p;n\}$, where b stands for "paying for a validated request in the future" and n for "not paying for a validated request in the future"
- $s_{\text{server}} = \{s;d\}$ where s means that the server will "serve requests" and d means that the server "does'nt server request".

Clients can decide if they are going to make another request later on that will be paid for. That means for the server that it can put the cost he had from the unvalidated request on top of the validated request. If the client chooses not to do so, the server will have been left with the costs created by the unvalidated request. It is assumed for the payoffs that sending a request and not sending a request have a cost close to 0. When the server is not serving a request it has no wins or losses. If he serves a request and the client is later buying a validated request he will have the cost of the unvalidated request but will also gain a payment k later on. We assume that $k > y > x$ where k is the value of the payment, y is the cost of the validated request and x is the cost of the unvalidated request. Thus the payoffs for the server are very simple:

- $u_{\text{server}}\{p,s\} = -x - y + k$
- $u_{\text{server}}\{p,d\} = 0$
- $u_{\text{server}}\{n,s\} = -x$
- $u_{\text{server}}\{n,d\} = 0$

Payoffs for the client can be described as follows: In both cases where the server is not responding to his request the payoff is 0, $u_{\text{client}}\{p,d\} = u_{\text{client}}\{n,d\} = 0$. The value for a response from the server is dependent on its validity. For the ease of this use case it is assumed that there is a solution that will prevent the server from lying by making its payoffs close to negative infinity. Thus they are dominated by other strategies and can be left out. This means that if a client is going to buy a validated request later, his payoff for a response will be $\{p,s\} = v - k$, where v is the value of the validated request. For the case that the client is not buying an extra request the payoff is $\{n,s\} = w$. A rational client will only send a validated request when $v >= k$, since he would make a loss otherwise. It can be seen that a business will only happen when $v >= k >= y + x$ and we will take this as a requirement. Seeing that the server needs to cover the cost for the unvalidated request we can safely assume that $w < k$ and thus $w > v - k$.

With those payoffs the game can be represented in a payoff matrix shown in figure 4.1. Those payoffs show that the Nash equilibrium lies with the client choosing not to buy a validated request and the server not responding. Even worse, the strategy to pay in the future does not even dominate the strategy to not buying one in the first place. The reason for that is the fact that v is not required to be bigger than k and k also covers the cost for w. Only when $v > k$ the game turns into a coordination game.

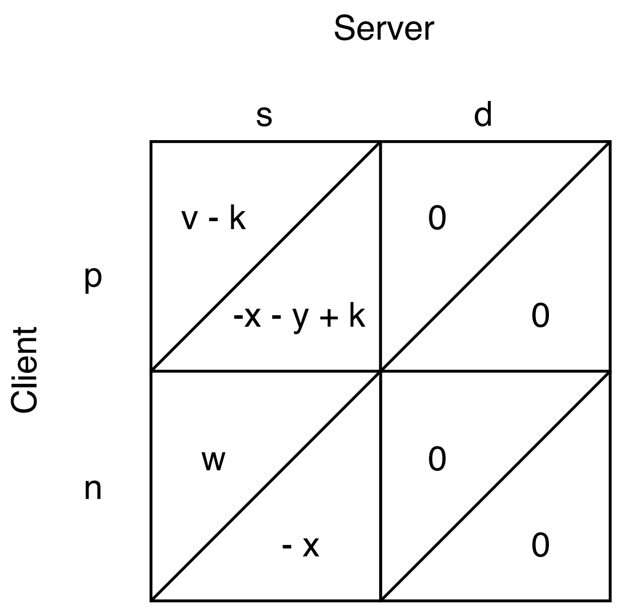

Figure 4.1: Payoff matrix

To make the protocol work it is necessary to change the payoffs for the play $\{p,s\}$ and make them more profitable for the client (or subsequently lower the profits for the play $\{n,s\}$) and lower or remove the negative payoff from the play $\{n,s\}$ for the server. There is one major restraint that if solved will do both and that is the problem that client identity is exchangeable. Under the assumption that it can be safely ensured that clients can be consistently identified, the game changes from a one shot game to a repeated game.

It is important to avoid unvalidated requests that are disproportionately sent to a sub set of servers. Therefore it must be ensured that all requests are distributed equally or that clients are forced to send a balanced subset of requests to a specific server. The first can be achieved by ensuring that server selection occurs in a fair distribution. The second can be achieved by forcing clients into playing a finite amount of games with a specific server and ensuring that the payoffs for the server are positive for the complete set of plays.

Server selection is covered in chapter 5, but it will not help with this particular problem. While it can suggest to the client what server to choose, it can not force it to do so and there is no incentive for the client to evenly distribute its requests. Selecting a server will suggest what the most profitable server for the client is which might not match with the goal of stabilising server cost.

The Problem with making clients play a finite amount of games is the fact that unvalidated requests come before validated requests. One way to solve this is to not only sell validated requests, but instead packages that contain unvalidated requests as well. Thus the client is strongly incentivised to use those requests but he is also not forced to use subsequent requests on the same server. This would also fulfil the requirement of identifying clients, since the server can give out a key that must be used in each communication. Package sizes can be adapted to the amount of requests the client projects to have in the future. Servers can freely set a price that could include price cuts if the client buys big enough packages or the server can make it more costly if only a small package is bought.

Similar solutions can be found that are already in productive use. API keys are distributed to clients that have different tier levels ranging from free to premium while free tier have a limited amount of free requests and higher tiers can be purchased. To avoid clients just generating free tier keys, those keys are often coupled with the underlying transport protocol, for example keys that are linked to specific ip addresses. Our solution provides a compromise by providing protocol level certainty of payment but also excludes anyone not willing to pay, thus raising the entrance barrier.

Strategies for the client then changes to buying a package or not buying a package in the first turn and then to simply sending a request or not. Payoffs change for the server to receive an initial big payment and only subsequent turns will then create the cost that will then be subtracted. As long as the wins from the initial package will cover those costs and the package price is not higher than the clients utility the game stabilises and both parties get what they want.

4.2.3 Peer to peer solution

A totally different approach to the original problem is to redefine the common to represent not the server capacities but instead the client requests. Server would then become the consumer. In order to sustain the system, servers would need to serve unvalidated request to prevent the network from collapsing. If no unvalidated requests are served, validated requests will stop and the common would collapse. It is now up to the server to coordinate the consumption of requests, by creating a peer to peer network for server nodes to fulfil the requirements as set forth by Ostroms for self governance.

In a peer to peer network, the nodes can punish each other by not forwarding any data. A reputation system can then represent the social structure. Ardabili and Liu [13] have shown through mechanism design "that not only a network has the incentive to provide information about itself [...] but also that this information can help decrease the estimation error". With that reputation system, servers can forward requests that they are not able to handle themselves to other servers based on their reputation score. If the criteria is set based on the amount of unvalidated requests served and the amount of validated request forwarded to others, the network can self regulate and find the optimal balance.

One issue that arises however, is that the client is not in direct control anymore on who will actually serve the request. It would simply send a request into the network and get a response. However, if the servers will keep track of performance within the reputation system it could even solve that problem. This kind of system would work great for requests that require a reasonable amount of computation since the overhead of forwarding messages becomes rather big. IN3 has not many requests that actually need much computation, thus this solution in its current concept is not applicable at the moment.

4.3 IN3's reality

While it has been shown that there exists a problem on a protocol layer, there where also a few assumptions made in the beginning that must be met in order for this system to justify the overhead. For one it was assumed that there is a large amount of request of the form of unvalidated. It was also assumed that the system runs over capacity, meaning the common could be over used. Both assumptions are correct in theory, in practice however it is far from true at the time of writing. Currently IN3 mainly supports Ethereum. IN3 only has one unvalidated request for Ethereum at the moment which is the blockhash. Validated requests are only partially dependent on that information. Also servers do not require a huge amount of effort to provide this information, since it can just cache it and only has to change it roughly every 10 to 19 seconds.

On the other hand, finding a solution to this problem is in a sense important for IN3 plans to support more blockchains in the future where that might be different. The other point is that the system is in fact overused. It could become a problem in the future but is unlikely if there is a proper incentive in place which is described in another chapter.

Lastly to mention is the attack vector of sending all unvalidated request to a specific subset of Server thus creating only cost and no gains for that specific set. While it does not produce any particular profit for the client, it certainly creates a problem for the server. Server selection solutions discussed in the next chapter do also directly fail in finding a solution to the problem, since attackers could just make their own selection. Attacking a server in such a way will only be successful however if a larger part of the clients do so as well.

5 Server scoring and selection

Since IN3 is not built as a network but more like a set of servers to choose from, the client must determine who the best communication partner is. For this it will run an internal scoring system based on meta data collected through its lifetime or past down from a superior host system. Internal scoring is made of preferences and develops depending on mostly external factors while still allowing the user to choose specific limits on chosen attributes. This chapter will discuss what factors play a role in the scoring and how it can affect the overall network performance.

As stated, the goal of the server scoring is to pick the best server to communicate with. Server scoring is thus a big part of the server selection process. Clients are not reliant on the settings installed by the protocol designer, hence only suggestions can be made here. However, the reasoning behind these settings is to have the best server handling requests. Clients are naturally inclined to pick the server that suits them the best but not those that are the most healthy for the system.

Taking a server with an optimal score, all clients will try to access that server. However the capacities for that server will be reached at one point and two things will happen. First, thinking about a rational acting server it would increase its price since the demand is higher than the supply. Secondly, all clients will start to downgrade the server internally because its response time went down and the prices went up. Servers that might have higher costs or that are newer to the system could have a better chance of being chosen. Internal scoring on the client side will thus not only allow the clients to use the best suited servers but also balance the load among the servers that meet the scoring criteria the best.

There are 5 scoring attributes that impacts the decision making process the most. These attributes are highly dependent on the business requirements.

Response Time This attribute indirectly measures server performance based on a set of different factors. Most prominently is the kind of request made. Some are more computation intensive than others and the scoring must take that into consideration. Multiple approaches can be taken here. Firstly the scoring could occur for each request individually. That would allow for a very detailed server selection based on the request the client wants to send. Servers could then specialise on a specific set of requests. Network performance could increase overall since servers with lower calculation capacity can serve low effort requests more easily and clients will quickly learn that requests that require more power should be directed elsewhere. It is even possible for servers to categorically reject certain requests without lowering its score perception overall but instead only on a request specific score. One disadvantage of that method is the clients capacities to store that amount of data. For each server not one but multiple scores must be stored, which might become an issue for IOT devices. An alternative is to take a specific request and measure the average. What request is chosen should be left up to the client to decide, if necessary even at random. This prevents servers from optimising just that one request since their score is measured against random requests. It is also possible to normalise each request. Thus the client does not need to store one list of response times per request for each server but instead only one general table with an average response time per request. This average would be taken from an initial probing and all following requests must measure up against that number. Since all response times are measured with the same unit they will affect the score equally[2] . If the client notices that most servers are strongly above or below the average it must correct its values from time to time.

Cost Measuring the cost has actually the same issue as response time in that each request is probably going to be charged differently. Thus either each server gets a price list for each request or a general table is used with an average for all servers.

Reliability In the context of IN3 we will define reliable twofold, one as measurement of ignorance and two as a measurement of dishonesty. For a client it is of importance to recognise servers that do not tell the truth. Servers can lie in multiple ways such as sending false answers to unvalidated requests or wrong data in the merkle proof. The server could also purposefully withhold information. Clients can not differentiate between lost information and withheld information. A Lie is for the client equally damaging as withheld information, thus we measure both in the same category. Lies will lower the score more dramatically as opposed to lost information. It can be argued that even false information does not necessarily originate from the servers intention to lie but instead from an error in the transport layer. Since transport layers however have their own way of detecting those errors we will assume that error correction will also be handled on that layer and the information that finally reaches the client has the intended content.

Age This attribute indicates how long the client has been dealing with this server already. The more often that server has been chosen to be used, the more weight does the score of response time for example have. If two servers have a response time of 5ms, one having served 1000 requests to the client and the other merely 10, the server that served the 1000 requests showed that he can sustain such performance over time, while the other that only server 10 might have only done so for those first ten. With age we can factor in the value of a long standing relationship and the security that comes with it. A Server that the client has communicated with for thousands of requests and that did never send a false response is way more desirable in some instances then for example more cheaper requests from a server the client never dealt with.

Deposit size For validated transactions, the deposit size plays a big role. It represents the security guarantees given to the client. In the event that a server is lying on a validated request, the client can slash the deposit as collateral. The higher the deposit is, the more secure the client can feel. This does not however have any effect on the unvalidated requests. Deposit size should then only be counted for where necessary. Important to note at this point is as well that this attribute is most likely going to be

[2] server1.responseTimeScore += requestType1.avg - response.responseTime;

related to cost. Servers with a high deposit represent a service for high value requests and can thus charge a higher price since it only aims at a specific part of the market, namely high value requests where a higher transaction cost is not of much importance as they do not appear as often as low value transactions.

More attributes can be added later during production to increase accuracy. For that however, more real life data is needed. In order to make a more accurate model, an additional factor has to be considered. This is because scoring is based on individual clients and it is important to ensure that the server that has just joined the list of in3 nodes will also be chosen. Clients act selfishly and if they already have a good sub set of servers they might not want to take the risk with a newer server that might be potentially more slow.

On the other side is however the fact that the new server might also be faster than the server in the clients chosen subset. This problem is known as an optimal stopping problem. The client has to decide at what point it stops looking for a better server. Optimal Stopping problems are often explained with the example of the secretary problem. Having multiple applicants, the task is to pick the best one out of all of them. However, the interviewing process costs time. Thus the question arises on when one applicant should be chosen.

The solution is to stop looking at the point when the probability of having found the best candidate is maximised. Problems of that category can be solved with the odds-algoritm shown by Thomas Bruss [7]. In short it shows a lower bound of 1/e, which is around 37%, meaning that the search should stop when the next best candidate is found after the initial 37% have been interviewed. In our example however we are trying to find the best sub set of servers from all servers on the node list. Bruss at al [8] showed in a follow up paper a solution for finding that subset. It shows an algorithm with which a set of best servers (observed records) can be selected with the highest probability.

With this algorithm, it can be certain that the maximum probability of the best subset of servers is chosen without having to iterate over each server on the list. In order to ensure that all new servers can be selected by older clients, the average score between servers in the subset is used. If one server goes below the threshold the client will remove it from the sub set and find another server that meets or exceeds the set average threshold. Additionally we make the sub set size dependent on the overall size of the node list. This allows for removal of servers if the nodelist shrinks or the ability to add to the sublist if the nodelist grows. The initially set server score average will deteriorate dependent on the fluidity of the nodelist. It is reasonable to run the search for the server sub set from scratch if a high enough number of servers on the list has changed or a significant amount has been added.

Those computations can be very taxing on the host machine because they are highly dependable on the random server selection and when a good observation is a record. Costs and duration of finding an optimal subset can thus be higher than the gain aimed for by the client. Clients themself would have no other choice but to run a sub optimal server selection. As a solution to that conundrum an external service could be created that does the server scoring and provides that information upon request. It would centralise an important part of the system's stability, but with enough independent sources the client could end up saving on request cost in the long run. Big companies running client fleets are more likely to implement such a service since the saved cost from running optimal requests is more likely to be higher than the cost created by running that service. In addition they would not have an additional trust issue between client and the external server scoring service. Services such as these would act as traffic regulators which would highly benefit the stability of the system as a whole and ensure that new servers have equal chance of being selected if their performance is up to market standards.

5.1 Validity estimation of unvalidated requests

Since there are requests that can not be validated from a single source, the client makes a decision based on probability. One option is to use its server scoring to determine trustworthy servers based on age and reliability. Secondly it can ask multiple servers for the same information and come to a conclusion on what the most probable truth is. It is also the first thing it can in fact do, because there is no server scoring in the beginning.

The easiest way to find a consensus is to take the most common response as truth. Dishonest or even just malfunctioning servers can be part of the servers that are being asked and thus we must calculate the probability that at least half of the requested servers have sent a wrong response. To simplify the calculations we assume that dishonest and malfunctioning servers are classed equally. Clients can not determine between dishonest and malfunctioning behaviour anyway which makes it also easier for them later on to calculate the percentage of problematic servers. Picking a subset of servers without picking any server twice is calculated through a hypergeometric distribution where

- N equals the number of server on the list
- M equals the number of dishonest server
- k equals the size of the subset being picked
- x equals the number of dishonest servers in the subset

The probability for x dishonest servers in a picked subset of size k is calculated by

$$P(X = x) = \frac{\binom{M}{x}\binom{N-M}{k-x}}{\binom{N}{k}}$$

If we want to know how probable it is that at least x number of dishonest server are in the subset we must sum the density for 0 up to x with

$$P(X \leq x) = \sum_{y=0}^{x} \frac{\binom{M}{y}\binom{N-M}{k-y}}{\binom{N}{k}}$$

.

The Number of servers on the list as well as the number of dishonest servers is not known in advance. Business requirements from the client are also important to determine the best size of the subset. For some requests the accepted probability of being cheated on is lower than for others. As a guideline figure 5.1 and 5.2 show the distributions for a list of the size 1000 with a chosen subset of 3 and 5 with a growing number of dishonest servers.

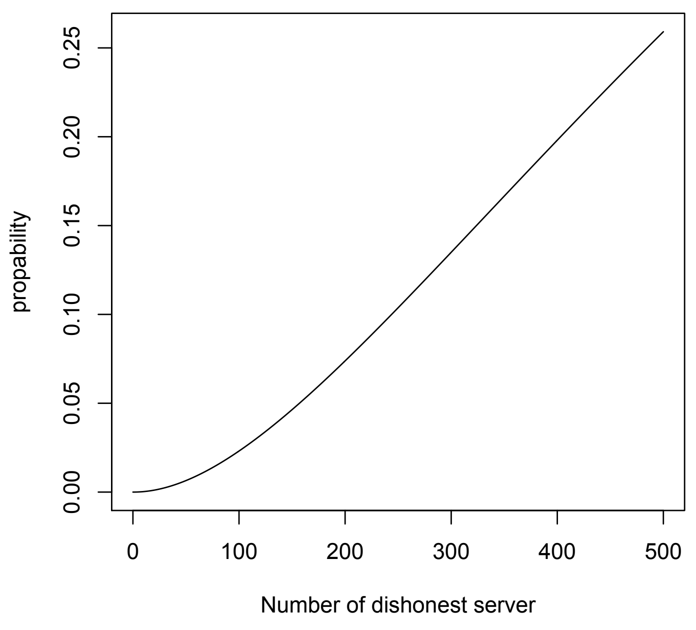

Figure 5.1: Probability distribution with N=1000, k=3, x>=2

It can be seen that clients must then decide if the cost of asking more servers is worth the added security. Appendix A includes some more graphs with multiple different parameters that might be realistic and can serve as a first guideline for implementation purposes. It is also important to note that the parameters are going to depend strongly on the server selection procedure, since it might exclude dishonest servers from the picking process done through the distribution calculations. Even with a simple and somewhat effective server selection, odds of being cheated upon can be lowered drastically. The client would start off by picking servers at random. With time it creates a list with servers that appear to tell the truth based on the experience it had. Servers that appear to send false information are scored lower and are then less likely to be in the base set of servers picked from (previously denoted as N).

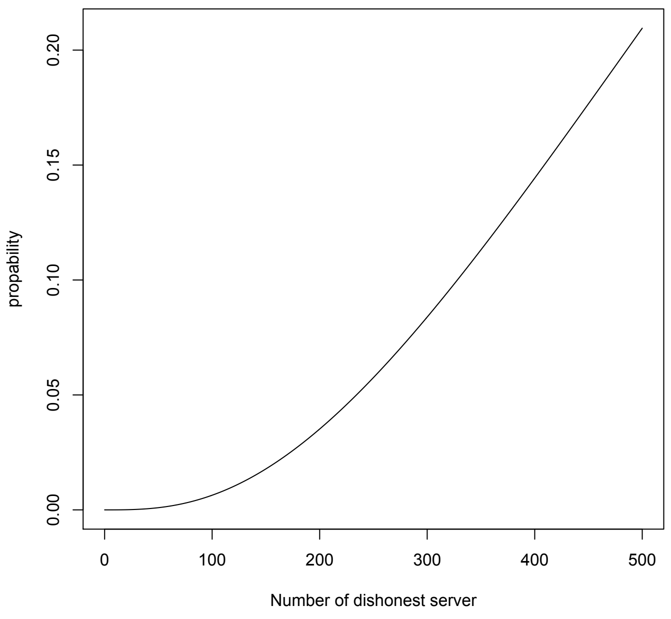

Figure 5.2: Probability distribution with N=1000, k=5, x>=3

6 Payments

One of IN3's main value propositions is decentralised blockchain access. Experience has shown that sustainability of decentralisation can be an issue. Ethereum for example was built around the idea that everyone is running an Ethereum node to access the blockchain. Each client would therefore be part of the p2p protocol. It didn't take long to realise that this is currently not feasible. Blockchain size increased too rapidly and soon smaller devices or even home computers could not keep up. Light clients were the first solution that tried to mitigate some of these shortcomings. The problem was and still is that light clients required real nodes to serve them data. Thus blockchain access was becoming more centralised.

Early ideas around light clients involved a token system that enabled full nodes to charge light clients for serving them. This is necessary since normal nodes do not have an incentive to spend resources on requests from strangers. IN3 is working in a similar fashion. It removes some of the restraints of light clients such as blockheader verification, but thus also moves it a step further away from the core technology. Light clients were developed together with full nodes and those core developer groups had easier access to Ethereum features. IN3 on the other hand exists on a layer further up.

Another solution that became big over time are remote nodes. Currently Infura is serving thousands of transactions daily. From a business perspective that doesn't make any sense, because there are no profits to be made. While they were first funded through outside investors they now started to take fees for their services. The main difference between IN3 and Infura is the fact that with IN3 certain requests can be cryptographically proven as valid while Infura must be seen as a trusted source. In order for IN3 to work, servers must be incentivised to take and respond to requests like Infura eventually did as well.

6.1 Non monetary incentives

One way to look at IN3 is from a customer's point of view. Companies want to use IN3 because they don't want to rely on a centralised or trusted service, even if it is their own. Having a distributed access network brings more security and reliability that can be marketed to the customer. In order for the network to have a sufficient capacity each company should run at least the amount of servers it is consuming as a whole. Additional security is added for their own clients in case all other servers in the system fail, as they would then limit the service of their server to their own client fleet. How do you guarantee now that the network will not be overused by companies that do not run their own servers.

Ignoring the results from the chapter on server selection it is assumed that that server selection occurs totally at random. Each client is then assigned a specific key that identifies them as being part of company a. A server from company n will, after a while have an estimation of the amount of clients each company holds, since given enough time, each client will pick that server n at least once. With that estimation the server can know if the amount of servers registered in the node list under company a's name are sufficient to cover all their clients. If that is not the case, company n's server will proportionally decline requests from company a's clients. Each company is thus incentivised to run at least the amount of servers to cover their own clients which results in an on average balanced network. Clients without a company can join and build partnerships or pay external companies to use their key. This would avoid the free rider problem described in chapter 4.1.

This solution has some issues. One problem is that the node list can not verify the true capacity a server holds. To circumvent that, clients can report periodically to its company about the amount of requests it has made to what server. Using that metric instead allows a company to more closely judge another company's performance. To lower the burden on smaller clients such as iot devices, a company can instead run watchdogs or buffed up client hardware. The problem with those watchdogs is however that they can only approximate the result and additionally put a burden on the systems capacity. The fewer watchdogs a company runs, the more inaccurate the estimation gets but the more watchdogs are run the more overhead is created for the system as a whole.

Another issue is that companies can hide the real amount of clients they have. To do that, the server divides the node list and distributes the parts between all clients. Other servers will now only receive requests from part of the client pool and thus think that the company only needs a fraction of the real capacity. Since it is unknown in how many parts the node list was split, it is difficult to make counter measures such as running multiple servers under aliases that communicate with each other. Proxy servers can also camouflage a servers identity and forward requests to a totally different server. Servers will appear on the nodelist that are in actual fact not serving the request himself but instead just redirect them. System capacity is thus bloated and a shortage of capacity is the result.

6.2 Monetary incentives

In order to encourage people to run IN3 servers, we incentivise them with the ability to make a profit. this is done by creating a simple market on requests. Chapter 1 goes into details on the difference of unvalidated and validated requests, but for this chapter we will summarise this into request packets. The contents of this packet shall not be of concern at this point, only that they are uniform among all servers. A market will develop very naturally if we provide a way for the client to purchase those requests, since the server has costs that need to be covered and the client has a demand for requests and a willingness to pay.

6.2.1 Market analysis

It is difficult to ensure a healthy market development in advance. This subsection examines a general market description and then looks into how we can avoid certain market failures on a protocol level.

The market described here consists of clients as buyers and servers as sellers. Answers to responses are the commodities sold. Server scoring is the driving force of the market which makes it competitive. Servers are inclined to offer faster and cheaper responses to the client as they will rationally pick the best one to do business with. It is easy to see that the market will most likely partition itself based on different attributes. Large deposits for example are more likely to be used by clients for high value transactions where costs are not that important. Costs would be however the driving force for low value transactions without time constraints such as sensor data from an IOT device.

Servers can specialise on parts of the market depending on how the server scoring is going to be implemented. Particularly of interest for market failures are monopolies. Especially in crypto-economics, monopolies can be very destructive not only on the market but the underlying technology. The whole premise of blockchains is decentralisation. Without it block manipulation and censorship can go unchecked. In the case of mining pools we have seen how large sized mining pools broke themselves up on their own because they saw the potential problem that came from it. In the case of access providers like Infura however, a break up has yet to occur. For IN3 the same problem arises where unchecked server clusters could create such access monopolies. This will not only endanger a fair price building but also attack the underlying value proposition of the IN3 protocol itself.

Monopolies destroy that value by centralising all traffic which in turns create a trust problem. In an ideal market that would lower the demand curve as the value for rational buyers for requests would go down, but the case of Infura shows that this might not be the case at the moment. Rational actors do not choose the most trustless or decentralised approach but instead the cheapest. For IN3 use cases however we will assume that this is not the case and instead that trustlessness increases the value of requests. It is a very optimistic assumption, but IN3 is built for a future that becomes more reliable on blockchain technology as trust in central banks and governments decreases.

At least in theory we can see that under this assumption a natural inclination to avoid market providers that control a large portion of available servers arises. In practise however, that is hard to enforce. Governments manage monopolies through law and ensure consumer protection. Real world examples such as mining pools show that some will act out of conviction, but others might not do so without having an incentive to do so, be it monetary or constitutional. The current implementation of IN3 has no way to enforce laws, such as anti monopoly laws, even if they would exist. This is a side effect of decentralisation coupled with real world detachment. Assuming that there exists a monopoly that creates the supply on requests, all that needs to be done is to register the servers under different names which would hide the monopoly. Any attempt to counter that fails at the exchange institution where privacy is the big value proposition for them. Thus the clients have no way of knowing who is taking part in the monopoly and the monopoly can then increase the price to the economic monopoly level.

Monopolies form where entry barriers or operational costs are very high.

IN3 has currently a very low entry barrier which consists only in the registering in the smart contract and the lock up of a deposit. With a sufficiently smart server selection algorithm or an external entity that provides an optimal list of servers to pick from, new providers have a fair chance against monopolies. As they enter the market they can offer lower prices than the monopoly which will push the overall price back to the competitive level. As soon as the competitive price is hit, there will be a surplus on request capacity and providers will back out. Even if the monopoly would stay, as soon as they increase the price again, another competitor would move back into the market since the demand would flow to a lower price. As long as entry and exit costs stay low the market will be mostly stable.

Monopolies will then only be dangerous if they undercut the competitive price. If a big player enters the market who can match all the demand with a low price, other providers will tap out. It is very questionable however how such a monopoly could potentially operate on a profitable basis, because with its monopoly power it can only hide information which very rarely can lead to actual profits, thus causing the whole operation to run at a loss.

A similar problem like monopolies are cartels. Providers can create a price fixing cartel that distributes profits among cartel members. Joining a price cartel is then becoming the more profitable choice. The cartel can also fix the supply of request capacity. In theory it would allow cartel members to adjust server capacities set by the cartel to push the price above the competitive level. In practice, forming cartels is actually more difficult to do on an electronic market such as this one. Main reason is that it remains very hard to prove that individual cartel members follow the set capacities, since it is more profitable to run at full capacity while being part of said cartel. Cartels would thus fail in their goal to fix the supply since no check or enforcement [3] can be done. It can fix prices but like with the monopoly, joining the cartel would not be the rational decision anymore because server selection pushes the price back to the competitive level.

It can be seen that the most common reason to have market regulations through governments is not needed for this market under the assumption that decentralisation is valued by the consumer and server selection is sufficiently implemented. For the case where decentralisation is not valued high enough, it can be assumed that there exist other technologies that serve those use cases much more cost effective than IN3 does. Server selection and ease of entry into the market are the most important aspects of IN3 that protects from market failure.

6.2.2 Defining a market transaction

IN3 faces a number of challenges when looking at transactions. Even though it extends the functionality of Blockchain technologies, it lacks a number of important traits that make blockchain transactions secure. The most important is the lack of an intermediary. Blockchains are considered trustless, but in the actual sense of trust that is not fairly accurate since both participants place their trust in the blockchain itself. Transactions

[3] Methods of bullying or punishments outside the protocol such as DDOS attacks are out of the scope.

between them are mediated by the blockchain who acts as a trusted third party. When thinking about IN3 however, this trusted third party does not exist. Clients communicate directly with the servers. Trust is an important part of a healthy and stable market. Plenty of economic research covers that problem and all converge on some form of reputation or reputation chains [10]. Regal defines reputation as âĂIJthe amount of trust an agent has created for himself through interactions with other agents". Banks for example only function well because customers have a certain amount of trust that they have put in the bank based on their or others past experiences. Electronic markets face that issue largely because now one should be trusted on the internet. Ebay's success [9] for example is largely based on their digital reputation system that helps consumers to determine if the online seller can be trusted, even if he has a pseudonym and can not be met in person.

In the concrete case of client and server communication that trust does not inherently exist. Certain attributes of IN3 hinder also the building of reputation on a system wide level as explained in chapter 2. Clients must rely on their own history of interactions with a server. Additionally, clients have no way of showing that they have been cheated on in the payment layer as opposed to the validity of a request they made. Two approaches can be taken when designing a transaction between a client and a server. The prepaid approach lets the client pay first and the server will deliver his response upon payment. Postpay is then the payment after the delivery has been made. From the clients perspective it is known that the expected payoffs for prepaid depends highly on the probability or trust that a server will deliver the requested response. From a servers perspective, its payoffs are the highest when it is cheating on the client. When the payment behaviour is changed from prepaid to postpaid, the exact same outcome can be observed, just with flipped roles. In order to change the behaviour to being honest the payoffs have to change, which then in turn increase the amount of trust put into a seller or buyer. Choosing between prepaid and postpaid will then depend on the ease of changing those results.

Consider a postpay system where the level of trust needs to be increased. Since the client moves second, the trust must be increased in the client. Request value is set by the client and can therefore not be changed, thus the payoffs of paying the server must be changed to an amount higher than the price it is supposed to pay as fee or the payoff can be lowered to a negative value through for example punishment. Known solutions require however that the client's identity is known to the server throughout the current and future interactions. Since anonymity payoffs are based on the cost of changing identities that cost can then be increased and thereby also increase the cost of not paying for the request, meaning that making anonymity more costly then paying for the request incentivises the client to honesty [4]. The Cost of changing identity is zero for two reasons: one is that there is no cost of creating an identity and second because there is no value attached to the identity.

[4] Anonymity in IN3 is more or less not a feature but rather a side effect that stems from the technology it uses. Removing this trait or weakening it is therefore not a problem until business requirements demand otherwise.

Most internet based markets create costs for identity generation just high enough to make automation not profitable but low enough to not scare potential customers away. One approach is a membership fee but most commonly the cost is more of an indirect result of the registration process. Limiting identities or in that sense accounts to one per email for example creates a cost of generating a new email address. Those portals move the cost creation simply to another system that deals with identity creations such as google or facebook. Platforms like that actually spend money to ensure that accounts are linked to real human beings. Costs are created there by simple puzzle solving or time restraints and dynamic algorithms ensure that the newly created account shows traits of real human behaviour. In IN3 these cost barriers simply don't exist. Clients are identified only in some instances by their Ethereum accounts, and even those can not be validated to be controlled by real people and have nearly zero cost of creation. It is important to note here is that there is no visible monetary price to pay by the client, since that would make the barrier of entry much higher from a psychological point of view. A central registry on the blockchain would then be very ineffective as the costs are actually visible as a transaction fee. This is still to be proven however.

Value attachment for identities are solved by reputation itself. Dai wrote "In a reputation based market, each entity's reputation has three values. First is the present value of expected future profits, given the reputation (let's call it the operating value). Note that the entity's reputation allows him to make positive economic profits, because it makes him a price-maker to some extent. Second is the profit he could make if he threw away his reputation by cheating all of his customers (throw-away value). Third is the expected cost of recreating an equivalent reputation if he threw away his current one (replacement cost)." [2] IN3 servers do not check transaction histories in any meaningful way. Identity value can then be solved by implementing a scoring system on the server side. Servers would keep a transaction record for each client and enable certain features only if a threshold score is met. Even identity creation costs can then easily be solved, since the server is keeping track of each client anyway. Thus it can link the Ethereum identity to a third party service such as google, facebook or github. For big company fleets, this can be solved by linking individual identities to company identities. Clients can present a company identification that can be checked against the company's server.

Prepaid systems shift the trust towards the servers. The problems here are similar to the postpay system. There however, are some traits that make it more favourable. Firstly the amount of servers is much lower, making the tracking of any global values much easier. Currently servers are tracked through the node registry making the creation of an identity very expensive which in turn discourages a server to cheat on clients. An argument can be made that the server does in fact not need to change its identity very often, since clients can not effectively communicate with each other, thus making the amount of request not necessarily much lower for the server by being put on a blacklist by a smaller number of clients.

There are very simple implementations possible to circumvent this. A reputation manager could be implemented that gathers information on servers like a watchdog and clients could then consult this service to retrieve a list of trustworthy servers. The trust is then shifted from the server to the watchdog. Watchdogs would still need to find a way to generate enough reputation [5] but thinking about a more tightly knit system within a company, clients pertaining to that company can inquire of a company's internal watchdog who is trustworthy by default. This would have the same effect as clients within one company or organisation sharing scores they made of servers between each other. Since they have a much higher trust among each other this information can be seen as reliable. If a server now cheats a client, that server is not only losing revenue from one single client but from a whole subset. As it is unknown to the server how big those subsets can be, the projected loss maximises since all clients could be in the same subset. Even statistical counter measures are of no effect here because clients can regularly change identity, meaning that if a server recognises that a set of clients stopped requesting from it after it has cheated, that information can become useless as soon as those clients changed identity. It must therefore only be proven that linked subsets of clients exist to prove that the cost of cheating will exceed the wins.

There are a number of cryptographic approaches that can help solve this specific problem of transaction without actually using prepaid or postpaid but instead simultaneous interaction. One such set of solutions are known as simultaneous secret exchange schemes, where both parties send an encrypted message to each other and then exchange the key bit by bit. Those protocols require correctness, meaning that the validity of each bit can be verified, and fairness, meaning that both parties have approximately the same amount of computational effort at each step, given that both have equal computation power. Fairness is thus simply not given in the context of IN3.

Deng et all [11] introduced a protocol for certified emails that employs a trusted third party to ensure that a receipt of deliverance is sent. This solves an existing problem with payments on IN3, as cheating is no longer possible by withholding payment or request response. this is because both are derived at the same time. The effects on performance however would be big, because each single request turns in to multiples based on the protocol implementation. It has also been suggested that the postman can be run within a trusted computation device. That would remove the third party as such and instead simply make the protocol requirements to run such a device on each side.

A third solution is using a protocol called coin ripping [12]. The general idea of the protocol is that the client would rip a bank note in half and send one half of it together with the request to the server upon which the server will deliver the request response to the client who will then send the other half of the bank note back. If neither of them cooperates, then the server will have lost the effort put into serving the request and the client will have lost the total amount of payment. Details of that protocol reveal however that in order for this to work, a trusted bank is needed that stops the client from spending his half bank note anyway (since this thesis is talking about digital coins, the client will still have the other half, thus the server would actually receive a encrypted bill that he then presents to the bank, indicating that the client is not allowed to spend that

[5] It can be thought that this entity becomes a licensed entity under government watch.

exact bill, and only accepts the bill from the server if he has the decryption key he would get from the client upon successful delivery).

6.3 Payment technologies

Many of the desirable attributes of IN3 must be conserved when picking a payment technology that drives market transactions, because unlike the underlying Blockchain technology, IN3 does not have that feature intrinsically built in. Additionally the chosen technology must be capable of serving hundreds of requests per second. Blockchain itself is already struggling to keep up with the demand and transaction prices rise across the board. IN3 is adding thousands of new users to the already heavily loaded blockchain. If for each IN3 transaction, even for those that do not in fact change state, another blockchain transaction is added on top and the resulting cost would be high and its capabilities would be maxed out quickly. Payment solutions that keep most of the value propositions of IN3 as much as possible are desirable.

The following section describes the two main Layer 2 scaling solutions: Sidechains and State channels. These solutions are the most researched and worked on ways to scale Blockchains up to serve thousands of transactions. Blockchains are not fast/scalable because they focus on security and decentralisation. Layer 2 solutions will either reduce decentralisation or security. Seeing that it is not possible to lower the security standards in payment solutions, there has to be a decline in decentralisation as a result. The following concepts will explain on a higher level how the sidechains and state channels work and what trade-offs come with them. This section will also look at some implementations that are ready to market and could be adopted as payment providers for the IN3 network.

6.3.1 Sidechains

Sidechains are moving much of the logic away from the main chain and load it onto another more centralised chain. This chain will provide another consent algorithm which includes fewer participants, therefore centralising the power. Crypto economic incentives or cryptographic proofs ensure that even though the system is centralised but still trustless. Exit strategies are put into place to allow the user to withdraw his deposited money even if the consent participants become malicious. Currently, there are two approaches that are the most promising, namely Plasma and Rollups.

Plasma Firstly there is plasma, which was developed in 2017 by the Ethereum foundation and is therefore well supported but also very complex. In Plasma users will lock up funds in a smart contract on the main chain. These funds are transferred to the plasma chain as soon as they are included in a plasma block. A set of validators is creating new blocks. Each commitment period the root hash of the merkle tree is stored on the main chain. Plasma comes with an elaborate exit scheme for the user to withdraw his funds from the Plasma chain back to the main chain. Every time plasma is implemented, new challenges appear which lead to many different versions of plasma like Plasma Cash which focuses on token transfers, Plasma Debit which adds payment channels to the logic of Plasma Cash and makes them transferable and Minimum Viable

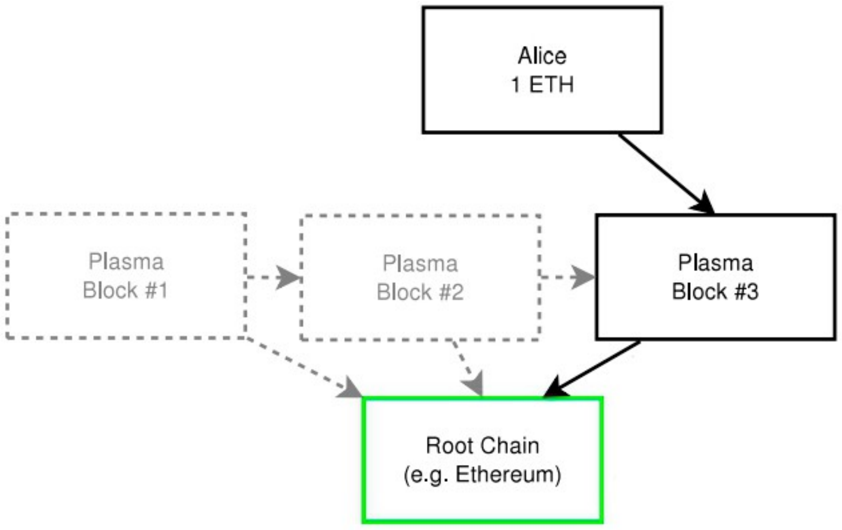

Figure 6.1: Plasma architecture [22]

Plasma which is a very simple UTXO based version of Plasma. All come with specific pros and cons [22].

Matic Network [14] implemented a variant of More Viable Plasma. They have a testnet deployed for ERC tokens and a ready to use wallet app is available. One problem with minimum viable plasma is that a congestion could happen if all plasma users want to exit at the same time. Because the linked Blockchain has only limited capacities and the exit must be done in a set period of time, it might come to a situation where so many users want to exit that the time to withdraw is too short. No solution for mass exits are implemented for Matic Network at time of writing. The software is run under the GPL3 License which makes integration much easier. Development is still ongoing and they have partnered with a few smaller blockchain startups. The Matic Network uses a dual strategy of Proof of Stake (PoS) at the checkpointing layer and Block Producers at the block producer layer to achieve faster blocktimes while ensuring a high degree of decentralisation by achieving finality on the main chains using the checkpoints and fraud proof mechanisms.

Basically, anyone can stake their Matic tokens on the root contract to become a staking operator in the PoS checkpointing layer (contract deployed on Ethereum chain). This provides a decentralised base layer for the Matic chain. At the blockchain layer of the Matic Network, there are Block Producers, selected by PoS Stakers on the base layer, who will be creating the Matic Blocks. To achieve faster block generation times, these Block Producers will be low in number. This layer is expected to achieve 1 second block generation times at extremely low to negligible transaction fees.

On Matic Network's checkpointing layer, the basis of Matic Network's PoS mechanism, for every few blocks on the block layer of the Matic Network, a proposer will be chosen among the stakeholders to propose a checkpoint on the main chain. These checkpoints are created by the proposer after validating all the blocks on the block layer of the Matic Network and creating the Merkle tree of the block hashes since

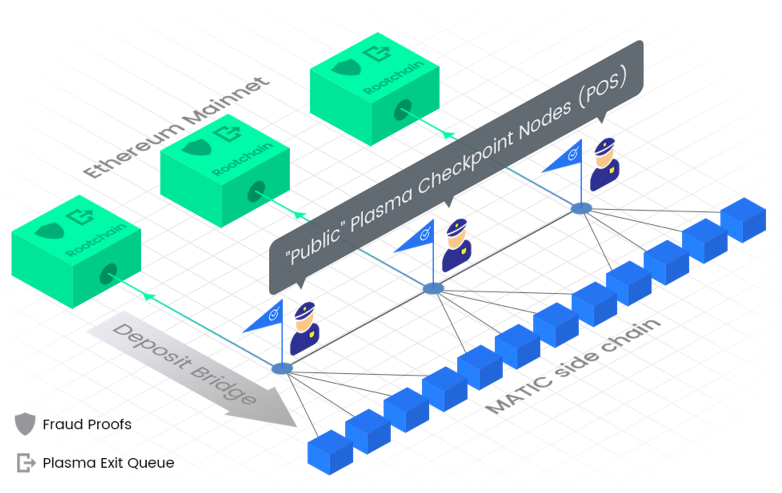

Figure 6.2: Matic Architecture [23]

the last checkpoint. The Merkle root is then broadcasted to the staker network for their signatures. The other stakeholders also verify the proof. They will approve the proposed block, if it is valid, by providing their signatures. The system needs the approval of 2/3 of the stakeholders to propose a "header block" to the root contract. Once the checkpoint is proposed on the mainchain, anyone on the Ethereum mainchain can challenge the proposed checkpoint within a specified period of time. If no one challenges it and the challenge period ends, the checkpoint is formally included as a valid checkpoint on the main chain.

Overall Plasma is a very well researched technology, but also a very complex one at that. Seeing that the Ethereum foundation has researched on that front for a long time now without one main solution solving all requirements for a second layer solution gives reason to worry. Matic Network's approach works but does not come with its own limitations. One of the main issues of Plasma is the long withdraw periods which makes that technology very user unfriendly.

Rollups The idea behind roll ups is to summarise a lot of transactions into a single one. That is achieved by generating an off chain block of data and only store a proof of it in the main chain contract. Two solutions exist that differentiate in one major aspect: data availability. In an Optimistic Rollup, the merkle root of the off chain state is stored in the rollup contract on the main chain. Every operator should verify the new merkle root, but it can only do so if the transaction data is available to them. Thus the term "optimistic" since they must rely on the truthful transition and the block producer must only generate a proof if someone is disagreeing with the transition. Block producers must also then make the data used for the transition available.

Zero Knowledge Rollups (zk-rollups) operate in the same way, except that instead of storing the data off chain, they generate a proof that the information was available at one point and store that proof together with the transaction data and the merkle proof on chain. One example would be that for each transaction a zero knowledge proof is generated that proves together with the transaction data that a valid signature existed for that specific transaction. Since the signature, the biggest portion of an Ethereum transaction, is then replaced with a much smaller proof, the block operator can then store the remaining transaction data (such as amount and receiver) in the call data (much cheaper place then the state) and writes the new merkle root into the smart contract. It scales Ethereum transactions to 10-20 fold depending on how much transaction data can be fitted into a single transaction. Users do not have to verify the state transition because the smart contract is already doing that by verifying the zero knowledge proof of the state transition. One advantage of optimistic rollups is that it can scale higher because it does not need to store additional data on chain. However it creates a problem wherein users must rely on the block operator to provide the data necessary to proof the state transitions correctness. Zk-proofs do not scale as much but are safer because the block producer has no way of manipulating the state transition. Proof generation however takes a long time (currently 20min) which puts a big time restraint on finality.

Matter Labs [15] is working on ZK-rollups and is funded by the Ethereum foundation. They lay out their roadmap very clearly focusing mainly on user experience with the next step in development being privacy. Currently, there is a beta version deployed that supports eth and erc20 transfers. No specific tx/s measure is given but estimates are currently 100tx/s on a 5-second finality. The software is developed under the MIT License as well as the Apache License. Currently, they work a lot on the Zero Knowledge Proof implementation. The user is depositing from the Ethereum network using a transaction to the ZK Sync smart contract on the mainchain. He can receive funds from anyone that already has funds in the rollup.

All transactions in the ZK Sync network are authorised with a signature using a public-private key pair. Public keys and addresses can be derived from the private key. A Public address in the ZK Sync network is required to receive funds. For convenience, it is possible to use Ethereum private key to derive a ZK Sync private key. All operations inside the zk-rollup are arranged in blocks. After a network operator creates a block it gets published to Ethereum with a Commit transaction. When a block is committed, its state is not final. After a couple of minutes a zero knowledge proof for the correctness of this block is produced. This proof is published to Ethereum using a verify transaction and after which the state is considered final. Multiple succinct blocks can be committed without being verified.

There are two types of operations for the user within the zk-rollup: Priority operations and transactions. Priority operations are initiated from the Ethereum network using Ethereum transactions send to the rollup contract. For example, the user creates a deposit transaction to move funds from Ethereum to the zk-rollup. Priority operation can be identified with a numerical id or hash of the Ethereum transaction that created it. Priority operations can't be ignored by the network. If priority operations are not being handled by the operator, the rollup contract goes into an emergency mode where users can withdraw all their funds based on the last verified state. Zk-rollup transactions are created and signed off chain with the zk-sync key. They are delivered straight to the rollup operator who then includes it into the next available block.

One downside of this is that even though the operator can not manipulate the state transition, it can decide which transactions go into a block and which don't. Matter Labs is working on an operator consensus that enables the generation of transaction receipts. A transaction receipt here serves as a promise to include a transaction into a block. If the transaction is not included the user can slash a deposit made by the operator.

StarkWare [16] developed a solution that implemented an optimistic rollup. They founded a Data availability committee (DAC) that ensures the state transitions are correct. While they also send a proof on chains that proves the correctness of the state transition, they do not store the required data needed to check the transition itself. That data is stored by each member of the DAC, and each new state transition needs to be signed by a quorum of members for the contract to accept it. The software is currently used in production for trading applications and is closed source. Optimistic rollups have better privacy properties since the data is not accessible by anyone, making the trading patterns of users private to the outside.

6.3.2 State Channels

Another well known technology in the blockchain industry are state channels. The general premise is that two parties exchange a signed state transition off chain. This state transition can be executed on the state channel smart contract to take effect. Signatures of both participants are needed for that. Instead of executing the state transition, the participants can update the data structure by changing the contents and re signing it. The security of the transition is granted by the smart contract. The participants only exchange that data struct offchain until either the parties have finished transacting with each other or one side is not in agreement with a proposed transition.

Payment Channels are a simplified form of state channels. The content of the transition is simply the balances of participant 1 (Alice) and participant 2 (Bob). Both Alice and Bob lock up funds in the payment channel smart contract. The state transition they are exchanging off chain is a change in the distribution of those funds. As shown in , Alice and Bob start with 5 ETH in the smart contract. Alice then sends a state transition to Bob which consists of the new Balances (Alice 4 and Bob 6) together with Alice's signature. Bob can now sign that transition and execute an Ethereum transaction which will change the balances in the payment channel contract. Bob does not want to do that, because he knows he will continue to do business with Alice. Either Alice or Bob can at any time propose a new transition with different balances. As long as both sign that transition it can be executed on chain. A nonce in the transition data will ensure that always the latest transition will be executed.

This concept can be extended to form a network, if a third participant is introduced (Charlie). Alice wants to send funds to Charlie, but she has no funds locked up in a payment channel contract with him. She has however an active channel open with Bob and Bob has one with Charlie. Alice will now send a secret value to Charlie that is needed to use the transition on chain. Then she will send a transition to Bob that requires the same secret value. Bob then uses his channel with Charlie to send funds to him, also requiring the secret value Charlie already possesses. If Charlie now goes on chain to execute the transition he will ultimately reveal that secret which can be picked up by Bob to close the transition with Alice. This system can be used either as a Hub network, where one entity is in the center of a start network and all participants have an open channel with him or as a peer to peer network.

A hub requires a lot of liquidity that needs to be locked up. A peer to peer network has the problem of path finding between the participants. An adaptation to payment channel hubs are virtual channels. Here Alice and Charlie have an open channel to Bob. Instead of sending a transition for each transaction through Bob, Alice and Charlie lock up funds within their payment channels that are open to Bob. Bob will then approve the lock and Alice and Charlie can exchange new transitions based on the locked up funds. It is basically a payment channel on top of another payment channel, but it saves a lot of effort on Bob's side.

State channels are a bit more complex. Instead of only changing balances, they can change the state of a smart contract. They are application specific because they rely on the state channel contract to have the logic implemented to make the agreed upon transition. Generalised State Channels move that logic also off chain. Instead of having Alice and Bob agree on a transition in a contract, they actually agree on a complete state that can be deployed directly onto the chain. With this, new features or requirements can be introduced into the transaction.

Raiden [17] is the most prominent name for Ethereum state channels. They have worked on a solution for years now and it is the perfect example to show that even though the general concept is simple, the actual implementation comes with a lot of complexities. Raiden is still in its beta phase and they have only implemented payment channel networks.

Connext [18] built a generalised state channel hub. It allows Alice and Charlie to extend the payments with conditions that are stored in a smart contract and works as previously described. To ensure that user funds are withdrawable at all times the hub must pose collateral.

Perun [19] is currently an academic project developed jointly by the TU Darmstadt (Germany) and the University of Warsaw (Poland). It is partly founded by the German Research Foundation and the Polish National Science Center. The Perun paper itself only discusses multihop payment channels, and construction of multihop state channels and payment channels of length > 2 was described in a follow up paper. Perun's paradigm is that a dispute in a multihop virtual channel must be (trustlessly) resolved through the intermediary. More concretely, if there is an Alice-Bob metachannel through Ingrid with a lot of functionality instantiated, Alice can grief Ingrid by forcing her to play out all the moves in the metachannel.

6.3.3 Choosing a solution

Use cases play a huge role in determining the technology stack. It must be considered who is interacting with whom for what reason. That will also determine things like who is paying the transaction fees etc. The use case is deciding at the end which is the cheapest and most efficient solution. For example things like usability plays a major role for payment between end customers but only a minor role for machine to machine payments like IN3 incentivisation. Each solution requires a lock up of funds. The duration of how long funds are locked up influences the user base. If for example a lot of funds are locked up in different payment channels and the client needs to close channels in order to pay for opening others, which also implies costs. What if not enough funds are available to close a payment channel etc. All of this is not a problem if the target user has high liquidity, but for some IOT use cases this might not be the case.

IOT devices also put a big limit on complexity. Keeping track of a lot of payment channels might not be feasible for a small sensor. Also things like availability are very important. Solutions such as payment channels where the recipient must be online to receive the funds are less likely to find a lot of traction. Sidechain solutions might be then more reasonable since the recipient does not need to be online and only one deposit is required to interact with any participant in the system.

7 Conclusion

This thesis explores a few potential problems that exist in the IN3 protocol and provides solutions to avoid them. Unvalidated requests can be used to overuse capacities at least on paper. The problem was presented with economic game theory and possible solutions where crafted to change the payoffs of the game to reach a favourable equilibrium. One solution is to pack unvalidated requests with validated requests and sell them together. Servers will then always run at least cost neutral and clients have no reason to not use those requests. Another solution was at least touched on that suggested the possibility to create a peer to peer network between the servers. They would then self regulate the incoming traffic based on a network wide reputation system. Even though that problem exists in theory, it is currently unlikely that it will become a reality. Unvalidated requests are right now not as important in the Ethereum implementation because only the request for the current blockhash would fall into that category. However, future implementations of other Blockchains could have a different distribution. To use unvalidated requests the client sends the same request to multiple servers and compares the results. A hypergeometric probability distribution was calculated to show how likely it was to have picked more than half dishonest servers.

The next issue discussed was the problem of server scoring and selection. Clients must select from a list a server it wants to send his request to. Scoring each server is the most sensible way for the client to do that. Multiple attributes such as response time or reliability where described. It was discussed on how those attributes can be combined into one single score, it was however decided that it strongly depends on the clients business rules and use cases. Long node lists could become a problem for weak client systems such as IoT devices. The optimal stopping problem was discussed which describes the selection process of picking a sub set of servers from the list while maximising the probability of having chosen the best servers from the list. If this represents too much effort for clients, an approximation seemed viable. An external system was also mentioned that could do the server scoring of the complete list and periodically update the clients with that information. Especially for larger companies with a lot of clients that solution seems most viable.

Lastly different ways to make payments possible and how to avoid market failures on a protocol level were examined. Non monetary incentives to run IN3 servers appear not viable because of how IN3 is built. This can be solved by resorting to monetary incentives and shows how the process of server selection can prevent monopolies from forming. Further than this, market transactions were defined to look at post and prepaid options. Both options can be solved by implementing some form of reputation system.

Post pay, meaning the client pays after he received the answer to his request, requires the server to trust that the client will pay. Clients can change their identity with 0 additional cost. Paying the server will then become not the most profitable strategy because the client can just change its identity and thus avoid any persecution.

Servers can create a scoring system for the clients and only unlock certain features if they deem the client trustworthy. Clients that belong to bigger trustworthy companies could have a higher default score. It is now in the interest of the client to keep its identity for the advanced features.

Pre-pay can work if the client trusts the server. Here the server is a known entity with a cost that is linked to identity creation. Servers do still have more leverage. Blacklisting servers on the client side will not harm the server since there are much more clients than servers in the system. A server reputation score service would change that. Communication between clients that use this external service will make the server lose more than just one client if he cheats, but the whole cluster. Since the server does not know how many clients belong to a cluster, his losses must be considered very high. For him it is now more profitable to just behave honestly. This solution is especially effective for larger companies that deploy big IN3 clusters. Private clients can also organise, but must trust the external service.

Concluding, current layer 2 solutions were analysed and probed for the use in IN3. Because of the large amount of requests, very small valued transactions on a large scale must be supported. Depending on the business requirements, the use of payment channel hubs or roll-up systems is suggested.

Appendix A:

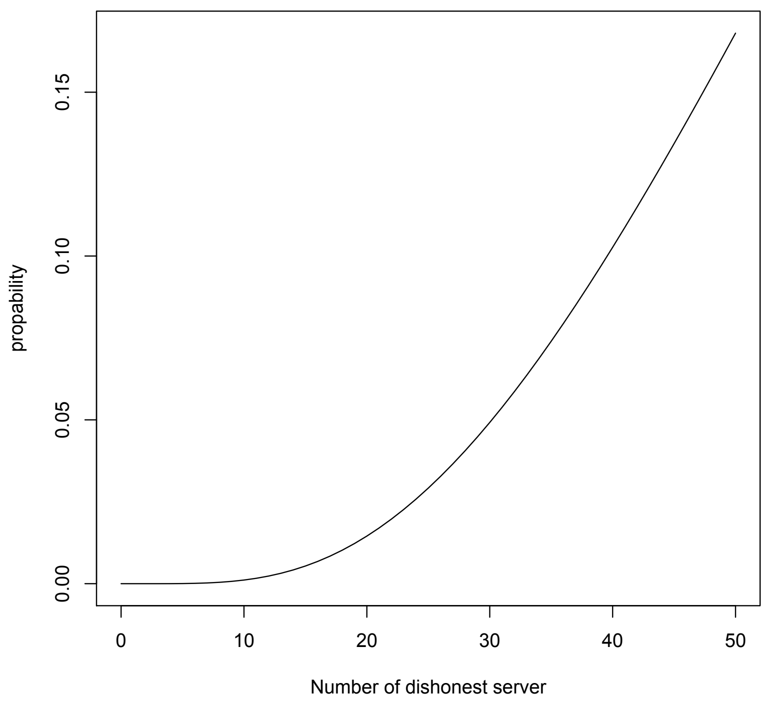

Figure A.1: Propability distribution with N=100, k=7, x>=4

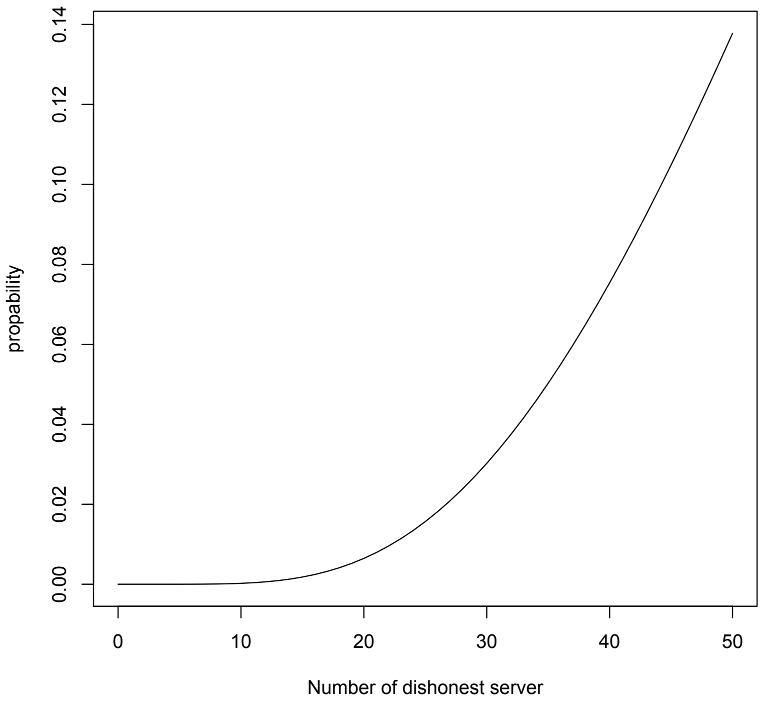

Figure A.2: Propability distribution with N=100, k=9, x>=5

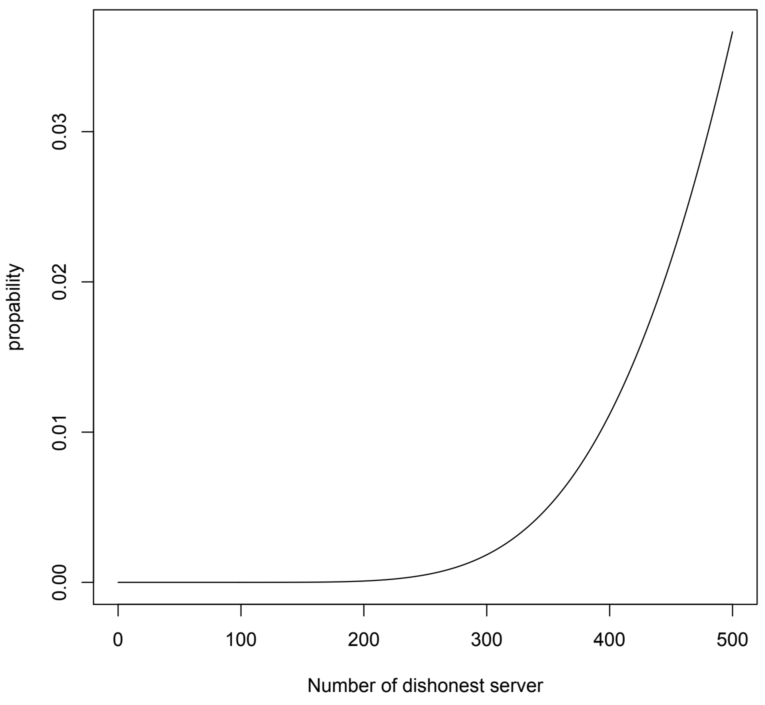

Figure A.3: Propability distribution with N=1000, k=20, x>=10

Bibliography

[1] Bentke, Jonas: Development and implementation of gas voting for the ethereum blockchain, 25 pages, 6 figures, Hochschule Mittweida, University of Applied Sciences, Faculty of Applied Computer and Life Sciences

[2] In H. Finney and W. Dai, 1995. "Re: Towards a Theory of Reputation," cypherpunks@toad.com: Tuesday, 21 Nov 1995 15:32:08 -0800. Cypherpunks' archives can be found at http://www.hks.net/cpunks/index.html.

[3] Hardin G. The tragedy of the commons. The population problem has no technical solution; it requires a fundamental extension in morality. Science. 1968 Dec 13;162(3859):1243-8. PMID: 5699198.

[4] Lloyd, William Foster. "W. F. Lloyd on the Checks to Population." Population and Development Review, vol. 6, no. 3, 1980, pp. 473-496. JSTOR, www.jstor.org/stable/1972412. Accessed 01 Oct. 2020.

[5] Peter Notbohm, 20 Liter Gratis-Benzin: Werbeaktion sorgt fuer Verkehrschaos, https://www.oberberg-aktuell.de/news/—liter-gratis-benzin–werbeaktion-sorgt-f–r-verkehrschaos-a-29703, 22.09.2020

[6] Elinor Ostrom et al. Revisiting the Commons: Local Lessons, Global Challenges. Science 09 Apr 1999: Vol. 284, Issue 5412, pp. 278-282 DOI: 10.1126/science.284.5412.278

[7] Bruss, F. Thomas. Sum the odds to one and stop. Ann. Probab. 28 (2000), no. 3, 1384–1391. doi:10.1214/aop/1019160340. https://projecteuclid.org/euclid.aop/1019160340

[8] Bruss, F. Thomas and Paindaveine, Davy (2000): Selecting a sequence of last successes in independent trials. Published in: Journal of Applied Probability No. 37 (2000): pp. 389-399.

[9] Hui, Xiang & Saeedi, Maryam & Shen, Zeqian & Sundaresan, Neel. (2016). Reputation and Regulations: Evidence from eBay. Management Science. 62. 10.1287/mnsc.2015.2323.

[10] Paul Resnick, Ko Kuwabara, Richard Zeckhauser, and Eric Friedman. 2000. Reputation systems. Commun. ACM 43, 12 (Dec. 2000), 45-48. DOI:https://doi.org/10.1145/355112.355122

[11] Deng, Robert & Gong, Li & Lazar, Aurel & Wang, Weiguo. (1996). Practical Protocols For Certified Electronic Mail. Journal of Network and System Management. 4. 10.1007/BF02139147.

[12] Jakobsson M. (1995) Ripping Coins for a Fair Exchange. In: Guillou L.C., Quisquater JJ. (eds) Advances in Cryptology - EUROCRYPT '95. EUROCRYPT 1995. Lecture Notes in Computer Science, vol 921. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-49264-X_18

[13] Ardabili P.N., Liu M. (2012) Establishing Network Reputation via Mechanism Design. In: Krishnamurthy V., Zhao Q., Huang M., Wen Y. (eds) Game Theory for Networks. GameNets 2012. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 105. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-35582-0_4

[14] Matic Network, https://matic.network/

[15] Matter Labs, https://matter-labs.io/

[16] StartWare, https://starkware.co/

[17] Raiden Network, https://raiden.network/

[18] Arjun B, Connext P2P Micropayment Integration Overview, https://paper.dropbox.com/doc/Connext-P2P-Micropayment-Integration-Overview-XNUi0Ysn8PbwrY08sf5jE

[19] Perun Network, https://perun.network/

[20] https://www.rdocumentation.org/

[21] Brian Curran, What is Ethereum's Infura? Scalable Access to Ethereum and IPFS, https://blockonomi.com/ethereum-infura/, 06.02.2019

[22] Plasma, Ethereum Foundation, https://docs.ethhub.io/ethereum-roadmap/layer-2-scaling/plasma/

[23] Matic Architecture, https://docs.matic.network/docs/home/architecture/matic-architecture

[24] Ethereum Foundaction, https://github.com/ethereum/devp2p/blob/master/caps/les.md

Erklärung

Hiermit erkläre ich, dass ich meine Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die Arbeit noch nicht anderweitig für Prüfungszwecke vorgelegt habe.
Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Jonas Bentke

Mittweida, 30. 10 2020