Rewarding Relays for Decentralised NAT Traversal Using Smart Contracts

Navin V. Keizer
University College London
navin.keizer.15@ucl.ac.uk

Onur Ascigil
University College London
o.ascigil@ucl.ac.uk

Ioannis Psaras
University College London, Protocol Labs
i.psaras@ucl.ac.uk

George Pavlou
University College London
george.pavlou@ucl.ac.uk

ABSTRACT

Traversing NAT's remains a big issue in P2P networks, and many of the previously proposed solutions are incompatible with truly decentralised emerging applications. Such applications need a decentralised NAT traversal solution without trusted centralised servers.

In this paper we present a decentralised, relay-based NAT traversal system, where any reachable node is able to assist an unreachable node in NAT traversal. Smart contracts on the Ethereum blockchain are used to ensure fair rewards. Besides financial incentives, a reputation system based on transactions on-chain is used to mitigate against malicious behaviour, and guide peer discovery.

Evaluation of our system shows that a combination of historic performance metrics leads to an optimal scoring function, that the system takes little time to reach stability from inception, and that the system is resilient against various attacks. Implementation of the smart contract shows that the cost for participants is manageable.

CCS CONCEPTS

• **Networks** → *Peer-to-peer networks*; • **Security and privacy** → *Distributed systems security*.

KEYWORDS

Network Address Translator, Smart Contracts, Network Relays

1 INTRODUCTION

Peer-to-peer (P2P) networks have been popular in the design of various applications such as BitTorrent and Napster due to their decentralised nature and associated benefits. Recently there has been a new surge to decentralise the Internet through P2P-based systems such as distributed file systems (IPFS) [1] and cryptocurrencies (Bitcoin, Ethereum) [14, 21]. There are, however, a number of challenges and issues in P2P networking, mainly due to the incompatibility with the current Internet infrastructure which is tailored towards the client-server model.

One of the most prominent is the issue of unreachable peers, which can be caused by Network Address Translators (NAT), firewalls, reverse proxies and more (from here on simplified as just NAT's). In a P2P network, a node must be able to act as a client and a server at the same time. However, NAT's make it impossible to receive requests from the network if the node itself has not previously set up a connection with the other node. Estimations in the Bitcoin and IPFS networks suggest that respectively at least 86.8% and 52.2% of nodes are behind a NAT [6, 20].

Early research in P2P networks [3, 18, 19] produced a number of protocols such as STUN, TURN and hole punching, aimed to fix this problem. A large suite of protocols originated due to the lack of standardisation in NAT's, making protocols only suitable for a subset of NAT's. For example, early research showed hole punching to be successful 60% for TCP and 80% for UDP [19], leaving relay-based protocols as the only reliable option.

One of the main issues with the previously proposed solutions is the need for a set of centralised servers assisting the network by relaying messages or maintaining connections with various nodes. These servers are prone to numerous attacks [18], require complete trust in the central infrastructure, and can be situated far from the node using its services, thus incurring high latencies.

For many latency-sensitive applications, such as P2P video calls, P2P live video streaming (such as in P2P gaming), and blockchain based computation offloading for latency sensitive tasks [17], this would cause significant performance degradation.

In this paper, we propose a decentralised, relay-based NAT traversal system for P2P networks with a built-in reward system for relays who assist nodes in their communication. In this system, any reachable node in the P2P network can become a relay, which requires incentivisation of honest behaviour and stronger security measures, as these nodes are untrusted and potentially malicious. Our system is realised by a combination of *i)* dual-path routing, producing a *Proof of Timely Relay* (Section 4.1), which ensures a client of honest behaviour of relay nodes, and *ii)* smart contracts (Section 4.2) on the Ethereum blockchain, which ensure fair exchange for the work done by relays. Nodes are incentivised to behave properly in order to claim maximum rewards.

A reputation system (Section 4.3) is incorporated on the blockchain to protect nodes from potential malicious nodes, and increase the chance of being used as a relay when behaving properly. Our *reputation scoring policy* is determined locally based on historical transactions on the blockchain, and assists in the *peer discovery* process (Section 4.4). Our evaluation (Section 5) explores different reputation scoring policies and their ability to avoid contracting malicious relays, and shows that the reputation system is able to quickly stabilise from inception. We finally look at the cost overhead and performance of the smart contract implementation.

2 BACKGROUND

In this section we review the general concepts of NAT traversal, blockchains, and smart contracts.

2.1 NAT Traversal

NAT's perform both port and address translation at the boundaries of public and private networks, enabling a much larger number of nodes in a private network to be connected to the public Internet. In general, there are two cases when NAT's create issues in P2P network communication between two nodes: *i)* both nodes are

unreachable (behind a NAT), and *ii)* only one of the two nodes is unreachable. In the first case, connection in both ways will fail, whereas in the second case only the connection from the public node to the unreachable node will fail. We refer to [3, 18, 19] for further discussion of these issues.

Many protocols have been proposed to aid in NAT traversal (due to the lack of standardisation in NAT's), mostly based on centralised servers. Out of these, TURN relays entire connections for case *i)*, with high reliability but low relative performance. For case *ii)*, connection reversal is used to relay the initial connection after which a direct connection is set up between nodes. These centralised approaches suffer however from a number of associated issues. Besides the well studied security issues [18], they do not suit decentralised P2P networks, as they require central management and infrastructure. We argue that this is at odds with the aims of a truly open and decentralised network, as no entity in the network should have higher privileges than others, and raises the question who would pay and manage the infrastructure.

Libp2p [10] has proposed decentralised versions of STUN and TURN, where any node in the network can replace the centralised server to help discover public addresses and relay connections. This approach, however, lacks peer discovery, bootstrapping, and required security measures (required since nodes are intrinsically untrusted and relays can easily launch an attack).

2.2 Blockchain and Smart Contracts

In a blockchain network, itself a P2P network, nodes have a full view of the shared history (i.e. the chain of blocks of transactions). This chain is append-only, which avoids double spending. The so called "full nodes" can send and verify transactions in the network, and perform a computational puzzle in order to produce (mine) the next block, for which they are rewarded.

One useful extension of blockchains are smart contracts, which are scripts of code stored on the blockchain and are executable by nodes. In Ethereum, a contract can be defined by users in the Turing complete Solidity language and can be deployed on the blockchain by compiling to the Ethereum Virtual Machine, after which it is added to the chain under its own address, which is callable by nodes. Executing functions defined in the smart contract has an associated *gas cost*, which is proportional to the added load on the network. Gas is paid in Ether, the cryptocurrency used in Ethereum, and the amount depends on the time constraints of the transaction. Read-only contract calls do not incur additional costs.

In order to write scalable and user friendly smart contracts, the function calls need to be kept to a minimum, as they result in gas costs. For this reason off-chain payment channels with incremental micro-payments can be used, where users submit a deposit and send incremental payment receipts, which the receiver can retrieve in one go, thus only incurring a single transaction fee.

2.3 Related Work

Existing work has proposed incentives for relaying network traffic with rewards. Ghosh et al. [4] propose the use of TorCoins to reward users offering bandwidth for relaying of TOR connections. Goyal et al. [5] introduce a system for decentralised content delivery incorporating secure incentivisation in P2P networks. Content providers start smart contracts for a file they want to distribute, and pay relay peers for content delivery. Norton and Simanavicius [16] propose a smart contract based segment routing WAN system, where users provide or consume spare bandwidth, forming a network of segment routers, used to find less congested network paths. Our work is distinct from these as we reward based on QoS measured, incorporate a reputation system to prevent contracting malicious peers, and use smart contract micro-payments.

There has been plenty of work on decentralised reputation systems in the past [7, 8, 12]. Closer to our work, Dennis and Owenson [2] propose a blockchain-based reputation system, where clients locally enforce scoring policies. We extend this with the use of smart contracts and explore local scoring functions.

3 THREAT MODEL AND ASSUMPTIONS

We assume that a P2P network comprises two sets: reachable and unreachable nodes. Reachable nodes can be relays, while unreachable nodes need a relay to participate in the network. Reachable nodes are able to function as a client and a server simultaneously.

In this system, unreachable nodes can use reachable nodes as their relays. Both are assumed to be potentially malicious actors, but rational. A malicious actor is one who may exploit the system for monetary gain and information, but also a node which oversubscribes its limited (bandwidth) resources and is thus unable to provide the good service promised. We assume the latter to be more common, as there are no direct financial incentives for launching an attack, but there are for serving more nodes as a relay (oversubscription attack).

Because relays and their clients mutually distrust one another, we use smart contracts deployed on a blockchain to function as a trusted, mediating third-party. One drawback of our approach is the requirement of network synchrony (i.e. constant blockchain monitoring), for example to avoid a timeout event as an honest node (Section 4.2). We make a simplifying assumption that all nodes have a stable connection to the Ethereum blockchain (based on previous relays or as light nodes), and that in the majority of cases the network is resilient and additional latencies in communications due to network-related events are rare.

Our system aims to present a secure decentralised NAT traversal mechanism for P2P networks, where malicious attacks are irrational and malicious nodes are easily identified, while ensuring honest nodes a fair exchange of rewards for their resource usage. These goals are achieved through incorporating the following:

- Financial incentives through smart contracts to stimulate good behaviour and mitigate against malicious attacks.
- Reputation system on-chain to identify potential malicious nodes and predict the performance of a node.
- Peer discovery based on nearby nodes and their reputation score (computed locally from on-chain transaction history).

Besides security, the system should have an increased performance in the best case (when an honest nearby node is contracted) over centralised solutions. The system and smart contract should not add significant overhead (processing, network traffic, and cost) compared to the centralised solution either.

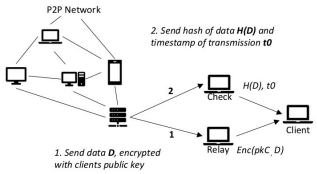

Figure 1: Proof of Timely Relay, data D is sent to the client.

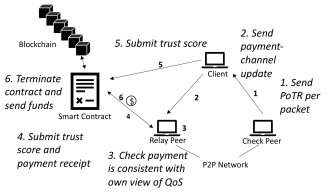

Figure 2: Relay operation and smart contract termination.

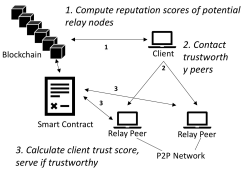

Figure 3: Peer discovery based on on-chain transactions.

4 SYSTEM DESIGN

The goals mentioned in Section 3 are reached through a combination of *Proof of Timely Relay* (PoTR), which guarantees the unreachable node that data has been relayed correctly and timely, and a smart contract which governs the rewards, ensures fair exchange, and implements a reputation system through mutual scoring.

When a client node contracts a number of relays to help become reachable in the network, an agreement is stored on the smart contract. Depending on the situation, the relay node either uses connection reversal (in the case the node dialling the unreachable node is reachable in the network) or sets up a circuit relay if both parties are unreachable. We focus on the latter since it has a higher reliability, and since we have no prior knowledge of network topology should assume the worst case. The mechanism is similar to that in the centralised case, but instead of using the central server, the relay node acts as the intermediary.

4.1 Proof of Timely Relay

PoTR, as shown in figure 1, is the mechanism through which a client can detect potential malicious behaviour by the relay. It allows the client, when communicating with another node, to verify that the data has been relayed without additional delay, data being altered, and a large packet drop rate, as promised in the contract.

In order for the client node c to verify this, they pick two relay nodes in the P2P network: one as a *check node* (R_c) and one as the *"true" relay node* (R_r). R_r acts as a standard relay, where a peer p (who is communicating with c) would send its data to R_r (possibly using end-to-end encryption to prevent tampering) which the relay node R_r in turn forwards it to c. At the same time, the peer p sends a hash of the data chunk and timestamp of the start of transmission to the check node R_c, which in turn forwards it to c. This information then allows c to assess the performance of R_r. Our simple PoTR mechanism can easily be extended, for example by sending a percentage of the data bits through R_c, as a small conceptual difference. We leave further exploration to future work.

c and R_r both keep a record of their view of R_r's performance in their local performance log, based on PoTR for c and based on forwarding delays and approximate network delays for R_r. Every evaluation interval (Section 4.2), both parties calculate the average performance over the interval they perceive and use this to settle cryptocurrency payments.

4.2 Smart Contract

A smart contract is used to document the agreement between clients and relays on the blockchain, and govern payments by c for the services provided by R_r without the need of a trusted third party. The smart contract allows for conditional payments based on the QoS provided by R_r, and uses micro-payments, ensuring fair exchange.

Our smart contract is implemented in Solidity and deployed on the Ethereum blockchain. When a client starts a new agreement with a relay, it submits the following negotiated parameters to the blockchain:

(1) Minimum QoS promise, as any statistic of: *i)* latency, *ii)* packet drop rate, *iii)* throughput
(2) Evaluation interval time
(3) Deposit
(4) Payment rate function parameters
(5) Timeout duration

The *minimum QoS promise* is the lowest QoS score that is acceptable for c for which it's willing to reward R_r. The QoS score is based on performance metrics such as latency, drop rate, and throughput. We envision this composition to be different for different applications, and leave the implementation to their developers.

The *evaluation interval* represents the time interval when c looks into its performance log and compares the interval's performance to what was promised (minimum) by R_r. If the observed performance is insufficient, c can terminate the agreement. Otherwise, c sends a signed payment update to R_r, according to the observed performance and the payment rate function (described below). A longer interval avoids the overhead of calculating metrics frequently, but this presents a trade-off with security as smaller intervals allow malicious behaviour on either side to be identified early.

Upon receiving the payment update, R_r compares this to what it expects (based on its own observed performance) to check that c is not underpaying. If it notices malicious behaviour from c or has other reasons to terminate the agreement (such as oversubscription) it can close the payment channel any time by submitting the latest signed payment update to the smart contract and triggering the termination process.

The termination process can be triggered by c or R_r to end the agreement and settle the payment. First, the node sends a termination message to the other, submits a trust score to the smart contract based on the perceived trustworthiness of the other node (explained in Section 4.3) and in the case of R_r also the payment receipt. Next, the other node submits the trust score they observed to the contract (and payment receipt if R_r), after which the payment is made to R_r

and the rest of the deposit is returned to c. The smart contract will only pay the funds when both trust scores have been received, and thus incentivises a terminating node to notify the other node, and incentivises both nodes to participate in the scoring. If one node becomes unresponsive and holds the other node hostage, a *timeout* event can be called (if the duration has expired) to settle the calling node's payment and automatically assign a negative trust score to the unresponsive node.

A client submits a *deposit* which will act as their relay credit, forming a pay-as-you-go structure. The deposit will set the duration of the contract, based on the maximum payment rate. A contract can be extended by additional deposits.

The *payment rate function* is the final parameter negotiated between c and R_r, and describes the payment rate for different levels of performance (QoS score). This incentivises R_r to perform better to receive a higher payment, and should be chosen to deter oversubscription attacks. For this, we propose a negative exponential function, but due to space limitations we leave the further details of the payment rate functions for the future work.

4.3 Reputation System

The smart contract records the trust scores given to nodes involved in the contract for each agreement. This makes for an open, decentralised and node-centric reputation system, as the blockchain is open and anyone can query the blockchain to get the previous transaction data, including the negotiated parameters (listed in Section 4.2) and the trust scores. Nodes can locally decide their own policy for calculating reputation scores of nodes in the network. This can be a pure indication of trust scores assigned in history, or go further and incorporate metrics such as contract duration and performance in terms of QoS scores. They can also opt to use public information only, or combine it with private information based on its own previous agreements.

The trust scores given to each node are either positive (i.e. 1), or negative (i.e. 0). We avoid re-join attacks by setting negative scores to 0. This way, malicious nodes will not gain from rejoining the network after receiving a negative score, as they will still start at 0 (the same as before).

4.4 Peer Discovery

In order for an unreachable client to discover potential relays, both the reputation of nodes and the expected performance are important. The reason we don't want malicious nodes to start with—even though they can easily be picked out in operation through PoTR and eliminated through the smart contract—is that they may still damage honest nodes. First, the time spent initiating a contract, waiting for an evaluation interval, and terminating a contract will delay the transmissions of files in the network and will reduce the overall QoS. Second, instantiating the smart contract and subsequent functions on it requires gas, which is paid in Ether, leading to a monetary loss. Last, packets might be lost due to the confusion, and this may affect the clients perceived trust by the network.

Contracting R_r with close proximity to c will likely lead to a better expected performance. Furthermore, the R_c would need to be close to ensure timely delivery of the PoTR. For this reason, peer discovery is proposed as follows.

First, c finds nodes providing relaying services nearby. There are various proposed ways this can be done, such as flooding the network and waiting for the first response, registering nodes at servers based on location, and using application level anycast [13]. Further details of this is left for future work. After obtaining the set of nearby nodes, c calculates their reputation scores, in order to filter out potential malicious nodes. Calculating only the scores of these nodes rather than all nodes in the network increases performance and decreases overhead.

Next, the top node N is contacted with a relay request, after which N locally calculates c's score to ensure it isn't malicious. If accepted, c submits all negotiated parameters to the smart contract, and if correct, N starts serving as R_r. We intend to extend this simple mechanism in future work.

5 EVALUATION

In this section we evaluate different aspects of the system by running various simulations, and show how our system meets the design goals outlined in Section 3.

5.1 Peer Discovery and Reputation System

In order to evaluate the performance of peer discovery using reputation metrics we used the PeerSim P2P simulator [11]. We created a network of 100,000 nodes, all with an ID and boolean variable to indicate maliciousness, thereby dividing the network into a set of malicious and honest, with all nodes acting both as a client and relay. We then generate historic transaction data on the blockchain for all nodes (number of previous transactions, trust, duration, and QoS scores) based on a Zipf distribution with different exponents for malicious and honest nodes.

We assume that most malicious nodes will start with a low number of historical transactions, because increase in usage would lead nodes to be more invested in the system due to built up reputation. A minority of malicious nodes will have a larger number of historic transactions (such as oversubscribed honest nodes), and we model this by setting the number of transactions using a higher exponent Zipf distribution for malicious nodes compared to honest nodes.

Every simulation cycle, each node randomly connects to k nodes (representing nearby nodes), performs a scoring function on their data, and chooses the best relay. A range of different scoring functions and parameters was tested in the simulation, beginning with scoring based only on the trust, QoS, or duration metrics. We also evaluate functions combining these metrics. For the rest of the simulation we set k to 5.

First, we measure the effectiveness of the different types of scoring policies made by the local nodes. Figure 4 shows the performance in terms of malicious node pick ratio *MNPR* (ratio of malicious nodes picked as a relay to the total nodes picked) for different malicious percentages in the network. The duration based and parameters combined scoring functions perform the best, followed by the trust based, QoS based, and random picking. Adjusting the ratios of the combined function we find a 1:4:5 ratio of QoS:trust:duration outperforms all strategies.

Next, we assess the impact of the number of historic transactions used in scoring. Figure 5 shows that, for the combined function, *MNPR* improves with a higher number of transactions used. There

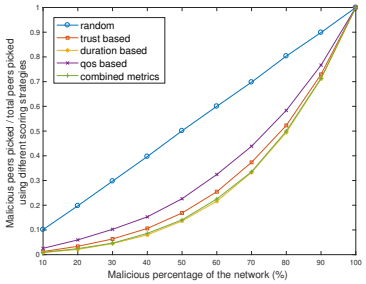

Figure 4: Malicious node pick ratio for different scoring functions (against malicious percentage in network)

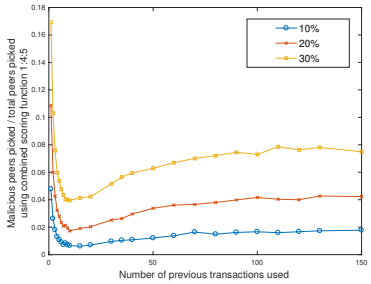

Figure 5: Malicious node pick ratio for different number of transactions used in 1:4:5 combined scoring

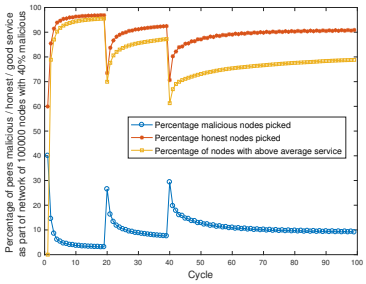

Figure 6: Stabilisation time from inception and after redistributing maliciousness at cycle 20 and 40

is however a trade-off in performance and precision to be considered, as for more than about 10-20 transactions used, the *MNPR* starts increasing again. This is due to the loss of accuracy of current behaviour as we average over a longer period.

Finally, we evaluate the time for the system to stabilise from inception. In order for our reputation system to work well in reducing *MNPR* transactions are needed on chain to indicate nodes ability to provide good service. However, at system inception all nodes start without any transaction data available and therefore we expect some delay in convergence to a stable reputation system. We require this period to be as small as possible, as users might be discouraged from using the system due to lower guarantees of avoiding malicious nodes.

For this simulation, we start with all nodes without any data, and have all choose the best relay out of their neighbours (based on the information available to them). Every simulation cycle, the picked relay is scored (based on malicious or honest) and is kept for the next cycle if their score is above average (set so all metrics are above 50%). If service was below satisfactory, the node chooses another neighbour as a relay. Nodes remember an unsatisfactory relay for one round. This avoids re-picking malicious nodes, but allows previously malicious nodes to recover and redeem themselves. We also redistribute the maliciousness of nodes at cycle 20 and 40, to observe if our system is able to quickly adapt and recover.

Figure 6 shows how our reputation system is able to stabilise quickly, and within 5 cycles reach a *MNPR* of under 10%. Furthermore, after some malicious and honest nodes switch at cycle 20 and 40, the system is able to stabilise within 10 cycles to slightly above the previous *MNPR*.

5.2 Security Analysis

We now evaluate the security of our proposed system and discuss how it mitigates against various attacks.

We first consider the situation when both c and R_r behave honestly, but due to network events, such as failures, the service provided is inadequate, leading to both nodes scoring negatively. Although this adversely impacts honest nodes, we assume that such network-level failure events will happen infrequently, and therefore this will not impact the overall trust score of a node significantly. Furthermore, our proposed combined metric scoring uses more metrics besides trust. In the negotiation phase, setting a low minimum service requirement also ensures that temporal network congestion does not immediately lead to contract termination, allowing R_r to recover and provide better service.

We next consider collude attacks between actors in the system. When R_r and c collude to try to boost scores with short forged transactions, they will not be able to impact the overall score by submitting trust scores, as the duration and QoS metrics will be controlled by the smart contract. Attempting this attack would also be expensive as every transaction on chain would incur costs. It is easier and more financially attractive to boost scores by serving nodes honestly, while earning rewards. Collision between R_r and R_c to keep c in the dark is also irrational as relaying packets properly would provide financial rewards, and sending dummy messages to c to maximise rewards will be noticed if this traffic is not expected, which makes this type of attack unfeasible as well.

Sybil attacks are largely ineffective as the attacker will have finite bandwidth. All new nodes entering the network start with a zero score, and will not be able to consistently serve as R_r unless they provide satisfactory service. Therefore, creating new identities or rejoining the network with new identities has no benefit for a malicious party. Using sybil identities to boost scores is also mitigated against since collude attacks are ineffective.

The final type of attack we discuss is a dishonest scoring attack, where nodes score dishonestly to lower competition in the network. This attack stems from the absence of direct loss when scoring dishonestly. However, when a node is scored incorrectly, it can keep the identity of the dishonest node in its log and avoid contracting them in the future. This can be shared with other nodes, potentially costing the malicious node future business. We leave further exploration of this mechanism and trade-off for future work.

5.3 Smart Contract

To evaluate the smart contract performance and costs, we implemented the code in Solidity and deploy it on our private virtual blockchain on Ganache. We couple the Remix IDE to interact with our accounts and functions. We obtain the costs associated with all functions based on the gas used, calculated using the GWEI/GAS estimates from the Ethereum Gas Station and taking the USD conversion rate in April 2020. Transaction speeds have varying costs, and should be used based on the time sensitivity of the function (e.g. submitting before a timeout event requires higher speed).

Function	Gas Cost	Cost slow	Cost medium	Cost fast
deploy contract	1738464	0.0086923 ($1.33861)	0.0132123 ($2.03469)	0.0173846 ($2.67723)
start relay agreement	231292	0.0011565 ($0.1781)	0.0017578 ($0.2707)	0.0023129 ($0.35619)
terminate agreement client	48440	0.0002422 ($0.0373)	0.0003681 ($0.05669)	0.0004844 ($0.0746)
terminate agreement relay	78980	0.0003949 ($0.06081)	0.0006002 ($0.09243)	0.0007898 ($0.12163)
timeout	62281	0.0003114 ($0.04796)	0.0004733 ($0.07289)	0.0006228 ($0.09591)
submit trust score client	65173	0.0003259 ($0.05019)	0.0004953 ($0.07628)	0.0006517 ($0.10036)
submit trust score relay	95970	0.0004799 ($0.0739)	0.0007294 ($0.11233)	0.0009597 ($0.14779)
extend relay agreement	34219	0.0001711 ($0.02635)	0.0002601 ($0.04006)	0.0003422 ($0.0527)

Table 1: Smart contract costs per function, with different transaction speeds

From table 1, we conclude that client costs of a contract would be around $0.40 - $0.60, and relay costs between $0.10 - $0.20. These costs may become significant if called frequently, but this is not expected as clients stay with a trusted relay over time.

Out of all contract functions, deployment is the most expensive. This should happen rarely however, as all agreements are stored on one contract. We envision contracts to be deployed per application using relays and the cost to be paid by the application developer.

6 CONCLUSION AND FUTURE WORK

In this paper we have presented a reward system for decentralised NAT traversal, where nodes are incentivised to be honest to earn rewards. To ensure fair exchange, off-chain micro-payments and a smart contract on Ethereum are used. Relays are rewarded proportionally to their performance, and clients are protected from malicious behaviour through a Proof of Timely Relay. A reputation system is used to discover nearby honest nodes, based on historic data on the blockchain, and nodes can enforce local scoring policies.

Our evaluation shows the performance of our reputation system for different scoring policies with different parameters, yielding a low malicious node pick ratio. Further, our reputation system is able to reach stability within a couple cycles of inception, and reacts well to nodes changing maliciousness. The costs of the interaction with the smart contract are reasonable for both client and relay.

In future work, we intend to further define the payment rate function and explore various system trade-offs, and extend our system so clients set up a pool of relays, which can dynamically be chosen for relaying, giving clients stronger guarantees of a backup if nodes are busy. We also intent to extend our system to other general application using relays such as decentralised content delivery.

ACKNOWLEDGMENTS

This work was partially supported by the EPSRC early career fellowship In-Network Service Provisioning (INSP) under grant agreement number EP/M003787/1.

REFERENCES

[1] Juan Benet. 2014. Ipfs-content addressed, versioned, p2p file system. (2014). https://arxiv.org/abs/1407.3561

[2] Richard Dennis and Gareth Huw Owenson. 2016. Rep on the roll: a peer to peer reputation system based on a rolling blockchain. *International Journal for Digital Society* 7, 1 (1 3 2016), 1123–1134.

[3] Bryan Ford, Pyda Srisuresh, and Dan Kegel. 2005. Peer-to-Peer Communication Across Network Address Translators. *Proceedings of the Annual Conference on USENIX Annual Technical Conference* (2005).

[4] Mainak Ghosh, Miles Richardson, Brian Ford, and Rob Jansen. 2014. A TorPath to TorCoin: Proof-of-Bandwidth Altcoins for Compensating Relays. *7th Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs)*.

[5] Prateesh Goyal, Ravi Netravali, Mohammad Alizadeh, and Hari Balakrishnan. 2019. Secure Incentivization for Decentralized Content Delivery. In *2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19)*. USENIX Association, Renton, WA.

[6] Sebastian Henningsen, Martin Florian, Sebastian Rust, and Björn Scheuermann. 2020. Mapping the Interplanetary Filesystem. https://arxiv.org/pdf/2002.07747.pdf

[7] Audun Jøsang, Roslan Ismail, and Colin Boyd. 2007. A Survey of Trust and Reputation Systems for Online Service Provision. *Decis. Support Syst.* 43, 2 (March 2007), 618–644.

[8] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. 2003. The Eigentrust Algorithm for Reputation Management in P2P Networks. In *Proceedings of the 12th International Conference on World Wide Web*. Association for Computing Machinery, New York, NY, USA, 640–651.

[9] Michal Król and Ioannis Psaras. 2018. SPOC: Secure Payments for Outsourced Computations. *CoRR* abs/1807.06462 (2018). http://arxiv.org/abs/1807.06462

[10] Protocol Labs. 2020. libp2p: Concepts. https://docs.libp2p.io/concepts/

[11] Alberto Montresor and Márk Jelasity. 2009. PeerSim: A Scalable P2P Simulator. In *Proc. of the 9th Int. Conference on Peer-to-Peer (P2P'09)*. Seattle, WA, 99–100.

[12] Tim Moreton and Andrew Twigg. 2003. Trading in Trust, Tokens, and Stamps. In *In Proc. of the First Workshop on Economics of Peer-to-Peer Systems*.

[13] Eleni Mykoniati, Lawrence Latif, Raul Landa, Ben Yang, Richard Clegg, David Griffin, and Miguel Rio. 2009. Distributed Overlay Anycast Tables Using Space Filling Curves. In *Proceedings of the 28th IEEE International Conference on Computer Communications Workshops, INFOCOM'09*. 19–24.

[14] Satoshi Nakamoto. 2009. Bitcoin: A peer-to-peer electronic cash system. http://www.bitcoin.org/bitcoin.pdf

[15] Till Neudecker and Hannes Hartenstein. 2018. Network Layer Aspects of Permissionless Blockchains. *IEEE Communications Surveys & Tutorials* (09 2018).

[16] William B. Norton and Jonas Simanavicius. 2019. A Blockchain-backed Internet Segment Routing WAN (SR-WAN).

[17] The Golem Project. 2016. Golem Whitepaper. https://golem.network/crowdfunding/Golemwhitepaper.pdf

[18] H. Schulzrinne, E. Marocco, and E. Ivov. 2010. *Security Issues and Solutions in Peer-to-Peer Systems for Realtime Communications*. RFC 5765.

[19] P. Srisuresh, B. Ford, and D. Kegel. 2008. *State of Peer-to-Peer (P2P) Communication across Network Address Translators (NATs)*. RFC 5128.

[20] Liang Wang and Ivan Pustogarov. 2017. Towards Better Understanding of Bitcoin Unreachable Peers. (2017). http://arxiv.org/abs/1709.06837

[21] Gavin Wood. 2014. Ethereum: A secure decentralised generalised transaction ledger.