Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed
Computing

Robust Scaling of Blockchain Protocols

Master's Thesis

Felix Laufenberg

`felixla@ethz.ch`

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Georgia Avarikioti, Yuyi Wang
Prof. Dr. Roger Wattenhofer

September 23, 2018

Acknowledgements

This thesis has a major overlap with two papers submitted and under review in the 23rd International Conference for Financial Cryptography and Data Security (FC2019). The papers titled "Towards Secure and Efficient Payment Channels" and "Short Paper: Designing Secure Sharding Protocols" are joint work with Georgia Avarikioti, Yuyi Wang, Jakub Sliwinski and Roger Wattenhofer.

Abstract

In this thesis we present vulnerabilities of state of the art scaling solutions for blockchain protocols, and provide novel protocols that improve on these issues. More precisely we propose a protocol (DCWC) that increases the security of payment channels and allows users to participate in payment channels without running a node. This is achieved by incentivizing witnesses to watch the channel and report potential fraud. Furthermore we show that state of the art sharding protocols provide insufficient security guarantees. We propose a model of properties, that a robust sharded transaction ledger should provide in order to give similar guarantees as a single robust transaction ledger such as Bitcoin. Furthermore, we provide a framework on designing a robust sharded transaction ledger and a protocol that achieves eventual consistency for transactions in sharded ledgers.

Contents

Chapter 1

Motivation

Increasing the transaction throughput of blockchain protocols without sacrificing security is arguably the biggest challenge for cryptocurrencies to effectively enter the retail market. Sharding protocols, such as [1, 2], attempt to increase the transaction throughput by processing multiple blocks in parallel. Sharding solves the scalability problem in general, making it a perfect solution for general purpose blockchains such as Ethereum [3]. However, these protocols are very complex and provide a large surface for possible vulnerabilities. Additionally there are still many open problems with making such a system work securely and efficiently. Layer 2 protocols such as payment channels and payment channel networks [4, 5, 6, 7] provide a simpler and possibly more secure solution to the scaling problem, by handling transactions off-chain. However unlike sharding, layer 2 solutions are not an all purpose scaling solution but rather solve the specific case of payment transactions. In this Thesis we discuss the state of the art and vulnerabilities of both scaling approaches and provide new protocols to improve on the state of the art solutions.

1.1 Payment Channels Introduction

Layer 2 protocols provide a simpler solution especially for payment transactions. Early versions [5] supported only unidirectional payments, but allowed these transactions to be confirmed instantly and off-chain, reducing the load of on-chain transactions. Duplex channels [7] allow for bidirectional transactions between two parties. More recent work [6] has improved the efficiency and usability of channels. Channels can be used to build a channel network using Hashed Timelock Contracts (HTLCs), as discussed in [4, 8, 9]. A payment channel network supports off-chain payments between users that do not share a direct channel. The Lightning network [4] on Bitcoin [10] and the Raiden network [11] on Ethereum [3] are implementations of such channel networks. Perun [12] supposes a new network structure, building around payment hubs in order to make channel networks more efficient. The authors of [12] also introduce a new model of channels that decrease the involvement of intermediaries for payments between parties that don't share a direct channel.

The security of channels relies on participants being responsive and running a full node. This is due to the fact that one party might try to settle on an old state of the channel, in which case the other party needs to publish a newer state during a specific time period. Thus, fraud can be proven and punished. Whoever can prove it must have access to a running full node to detect that somebody tried to settle on an old state. Running a full Bitcoin node is costly, i.e. it requires hardware that most end-users don't have at home and some skill to set up a node without significant downtime. The cost of renting hardware to run a node is unreasonably high just for making payments, and for the Lightning network [4] the fees collected for routing transactions seem not to cover the expenses of renting a node, as explored in [13], while this also depends on the liquidity and number of channels that a node has. Moreover, even if a party is running a full node, he might lose funds if his node crashes or goes offline for an extended period of time. The Bitcoin community has proposed using watchtowers [14] as third parties to watch channels, but to the best of our knowledge, incentivizing watchtowers remains an open problem. This is the first problem we address in this work.

In a simple watchtower solution, miners could act as watchtowers to keep track of all channels off-chain and publish a proof of fraud to collect fees. Such an approach faces various problems. Firstly, throughput; having all miners receive, store and send all off-chain payments leads to network congestion which in turn leads to lower throughput and recentralization as more powerful hardware is required to become a miner. Secondly, it is difficult to predict the miners' behavior in such a system since miners are not incentivized to propagate the proofs of fraud. On the contrary, it might be in the miner's best interest to keep the proofs to himself to increase his probability to collect the fees if he publishes a block. Our approach builds a network of watchtowers around each participant of the payment channel network. Watchtowers increase their expected payoff by faithfully executing the protocol, watching the blockchain and forwarding channel updates to other watchtowers. We prove that our protocol is secure and incentive-compatible, while only requiring watchtowers to store $O(1)$ messages. Concerning privacy, watchtowers do not learn about the distribution of funds between the involved parties. Furthermore, we present an adaptation of our protocol implementable on the Lightning network.

1.2 Sharding Protocols Introduction

Cross-chain systems exist whenever multiple blockchains interact with each other. They exist on top of the protocol layer, when we have interactions between heterogeneous blockchains as in [15], or sidechains [16]. Or they can be part of the protocol layer as in sharding [17, 18]. In either case, interoperability between multiple blockchains must be robust and inter-chain transactions should eventually either exist in all involved blockchains or in none of them.

Sharded protocols [17, 18] create a shared ledger, operated under multiple homogeneous blockchains. Sharding promises to be the solution to throughput limitations of blockchains. Increasing the throughput on a single blockchain increases the hardware requirements of miners and leads to centralization of the network as discussed in [19]. On the contrary, sharding has the potential to be a scale-out solution by reducing the hardware requirements of miners by splitting the ledger in multiple shards and allowing miners to only know about a single shard to participate.

1.2.1 Challenges in sharding

Recent advances in sharding research, mainly through the contribution of Omniledger [18] and the Ethereum community [17] were able to solve many of challenges in designing a robust sharded ledger. However, open challenges remain. A consistent model of properties for robust sharded transaction ledgers is missing. In [18] inter-shard transactions are robust under the condition that no stale blocks exist in the system. This is achieved using byzantine agreement (BA) to avoid forking. Network issues or powerful short term attacks can always lead to forks. Once a stale block occurs leading to an inconsistent inter-shard transaction, [18] is unable to recover.

Inter-shard transactions may be payment or contract transactions. For simplicity, let's investigate the case of a payment transaction from chain A to chain B. as pictured in 2.1. Block b_A of chain A includes the initiating transaction tx_A and block b_B the accepting transaction, including a proof that tx_A has been confirmed in A. If b_A turns out to be a stale block in chain A, but b_B remains an accepted block in chain B, then the two chains (or shards) A and B have an inconsistent view. In other words, while both chains individually appear consistent, the global consistency is broken

A constant probability of failure remains. This probability can be decreased almost arbitrarily by tweaking parameters, such as weakening the security model or decreasing performance. In either case, the protocol should provide a way to recover from such a scenario.

We present a model for robust sharded transaction ledgers. The model provides similar properties as a single robust transaction ledger as defined in [20] under similar assumptions. Additionally we provide a high level framework for designing such a ledger and propose a new protocol that allows sharding protocols to recover from stale blocks. The protocol can be incorporated into existing sharding protocols such as [18] or any protocol that deals with inter-chain communication.

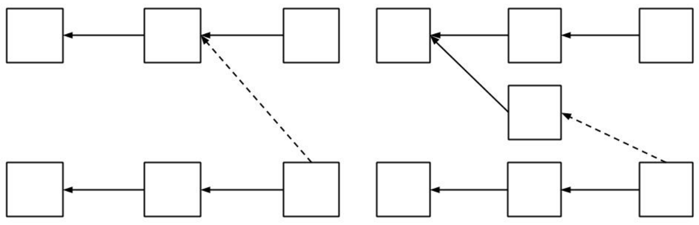

Figure 1.1: Inter-chain transaction without (left) and with (right) stale blocks.

Our contribution is summarized as follows: In Chapter 2, we present a novel protocol to secure channels by incentivizing watchtowers. We prove our protocol is secure and incentive compatible in Section 2.2, and present an alternative version implementable in Bitcoin's Lightning network in Section 2.3. Additionally, in Chapter 3 we propose a model for robust sharded transaction ledgers, propose a framework on how to construct such a ledger in Section 3.3 and introduce a new protocol that provides eventual consistency for sharding protocols in Section 3.4. We discuss related work in Chapter 4 and conclude with future work in Chapter 5.

Chapter 2

Incentivizing Payment Channel Watchtowers

There are many different models and implementations of channels such as [6, 7]. While the implementations (e.g., Raiden, Lightning) differ, the concept remains mostly the same. A channel protocol typically consist of three phases: The *setup* phase, opening a channel; the *negotiation* phase, handling off-chain transactions, and the *settlement* phase closing the channel and making outputs spendable. Our protocol builds on top of a channel protocol and is thus applicable to different implementations. To achieve better consistency in this paper, we rely on a simplified model of channels. On a high level: Two parties create on-chain funding transactions to start a channel, blocking their stake on the blockchain. Both parties sign subsequent off-chain update transactions representing the current distribution of stake amongst them. Either party can create a settlement transaction to close the channel. If party A issues an old settlement transaction and party B can prove it is old by presenting a update transaction signed by A with a higher sequence number, called proof of fraud, then B is awarded all stake in the channel. In our protocol the parties include rewards for miners and watchtowers in the funds of the channel.

Whenever an update transaction is issued, the party B who receives a payment wants to be sure that some part of the network knows about this update. A simple solution would be to spread the newest update transaction to as many nodes as possible and offer a reward for whoever sends a proof of fraud to the miners in case of a misuse. Such a protocol could accidentally or purposefully cause congestion at block miners, providing the possibility to launch cheap DDOS attacks. Nobody would know how many nodes have seen the update and are willing to send it to the miners. Also nodes would have no incentive to spread the contract as it would only decrease their chances of getting paid.

2.1 The DCWC (Disclose Cascade Watch Commit) Protocol

DCWC consist of l rounds. During each round a group of watchtowers gets the chance to send a proof-of-fraud and collect fees. Once a proof-of-fraud has been included on the blockchain or l rounds have passed, the protocol terminates. Each round consists of some predetermined number of blocks m. If no proof of fraud was published after l rounds, the settlement transaction will be accepted as the final state of the channel and the funds will be unlocked. The protocol terminates after $l \cdot b < t$, where t is the timelock off settlement transactions in current solutions [4, 7]. We improve the time that funds are locked after a channel is closed one sided, compared to state-of-the-art solutions. Intuitively, our protocol can terminate much earlier since many more entities are online and watching the channel. Note that the channel can be closed instantly if both involved parties are cooperative and online.

When a new channel is opened, both parties provide a fund on-chain for paying transaction fees for a proof of fraud and paying the watchtower that published it. The protocol consist of two phases. The first phase is executed once for every newly issued update transaction. It is responsible for spreading the update in the network. The second phase ensures that a proof of fraud will be published, when an old update transaction is published as a settlement transaction. The second phase consists of l rounds, where l is specified on the funding transaction and should depend on the amount of funds locked in the channel. Our protocol requires participating watchtowers to store only $O(1)$ messages as the message with the highest sequence number is a proof for all others. In terms of *privacy* watchtowers know which channel they are watching but they don't need to learn anything about the payments except for the sequence number.

2.1.1 Phase 1: Disclose & Cascade

The first phase is entered after the funding transaction has been included in the blockchain and terminates when a settlement transaction is included in a block. This phase ensures that a sufficient amount of watchtowers receive an update message, and also that these messages are correct. Essentially, participants send new update messages to their neighbours for a few iterations. These messages are cryptographically designed to capture the travelled path in order to reward watchtowers that forwarded messages. Disclose of Phase 1 is invoked by a party A/B, involved in the channel, anytime he or she receives a new update transaction $t_{c,i}$ with sequence number i. Cascade of Phase 1 is invoked by any node receiving a message generated from Disclose or Cascade. The parameter N limits the number of neighbors to whom a message can be sent. The value d_m

determines the number of hops that a message m has traveled. Let $\{m\}_K$ denote that m is encrypted with private key K.

Algorithm 1 Phase 1: Disclose & Cascade

```
 1: procedure Disclose
 2:     $K \leftarrow own\ private\ key$
 3:     for ($i \leftarrow 1$; $i \leq N$; $i \leftarrow i + 1$) do
 4:         $W \leftarrow find\ new\ neighbour$
 5:         $m \leftarrow \{i, W, t_{c,i}\}_K$
 6:         send($m$) to $W$
 7: procedure Cascade
 8:     $in \leftarrow receive\ message$
 9:     $t \leftarrow d_{in} + 1$
10:     if $l \leq t$ then return
11:     $K \leftarrow own\ private\ key$
12:     for ($i \leftarrow 1$; $i \leq N$; $i \leftarrow i + 1$) do
13:         $W \leftarrow find\ new\ neighbour$
14:         $m \leftarrow \{i, W, in\}_K$
15:         send($m$) to $W$
```

Thus,

$$\begin{aligned} t_{c,i} &:= \text{i-th update transaction of channel c} \\ m &:= \{id, W, m'\}_K\ or\ \{id, W, t_{c,i}\}_K \end{aligned} \tag{2.1}$$

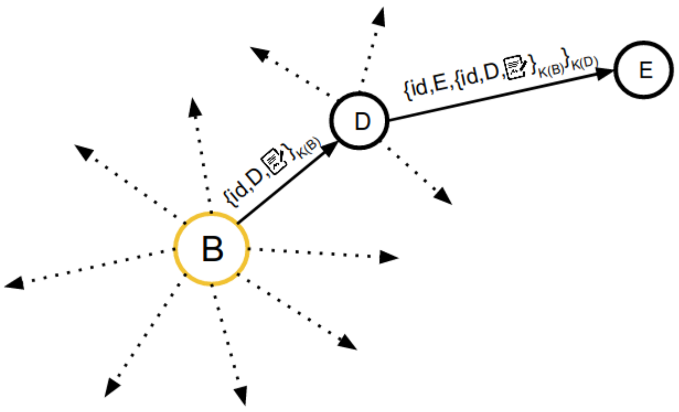

Figure 2.1: Participant B in the channel disclosing the newest update message to his neighbours. Watchtower D cascading the message to his neighbours, one of whom is watchtower E.

2.1.2 Phase 2: Watch & Commit

The second phase of the protocol ensures that in case of fraud, i.e. A or B publish an old update transaction, a proof of fraud will be published. Throughout the lifespan of the channel all watchtowers that are involved in the protocol watch the blockchain for such an update message (WATCH process). After a settlement transaction $t_{c,i}$ of a channel, signed by one of the parties involved in the channel, is published in a block the watchtowers start the COMMIT process. It consists of a fixed number of rounds l, specified in the channel. For simplicity, we assume a round corresponds to a single block. However, the number of blocks that determine a round can be adapted without affecting the protocol. We note that block frequency and propagation time should be considered to estimate the reaction time of the watchtowers, and thus the number of blocks per round. The actors in this phase are the watchtowers.

Let $t_{c,j}$ denote the newest update transaction that watchtower W has seen. Let m denote the message as created in the Disclose & Cascade phase through which W learned about the newest update transaction $t_{c,j}$. Watchtower W stores only this message and can discard any other message concerning channel c thus reducing his storage costs. After d_m blocks trail the settlement transaction $t_{c,i}$ and $j > i$ and no proof of fraud has been published yet, W signs and sends m to the blockchain network. With m, W also sends all information that m contains in plaintext. This information includes the path that the message has travelled, and the id at each hop of the path. The plaintext information will not be included on the blockchain, but helps the network to reduce the number of invalid messages that are send around. If the plaintext of the message does not match the encrypted information, then the network nodes will discard the message.

2.1.3 Payoffs

After l rounds have passed, the channel is closed. At this stage the funds of A and B are unlocked and the watchtowers are paid.

Ideally, all watchtowers would get pay their marginal contribution for participating in the protocol. However, a protocol that implements such a payment mechanism would require many on-chain payments to pay all involved watchtowers. Thus, we propose a payment mechanism that pays only the watchtowers that participated in forwarding and publishing the update message that has been included on-chain. When the channel is created, both parties involved in the channel reserve some value ρ_c for paying transaction fees and rewards for a proof-of-fraud. When a proof-of-fraud has been included in the blockchain, all watchtowers included in forwarding that proof-of-fraud get an equal share of ρ_c, reserved by the cheating party, after the transaction fee has been deducted. In

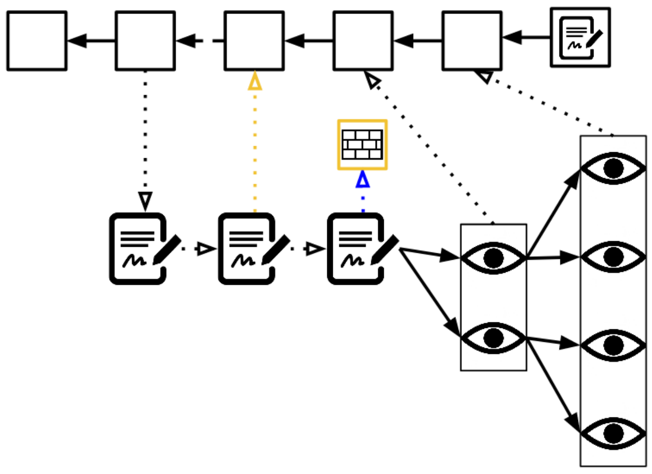

Figure 2.2: Vizualization of COMMIT process. Party A (yellow) publishes an old update as settlement transaction and prevents B (blue) from publishing a proof of fraud. The groups watchtowers, depicted as eyes, get the chance to publish a proof-of-fraud during the following rounds.

the following section, we show that this payment mechanism ensures sufficient expected payoff to incentivize the watchtowers.

2.1.4 Transaction validation

The miner of the next block checks whether the messages were generated according to the protocol. Out of the set of valid messages he or she selects one uniformly at random and publishes it in the new block. The miners perform the following checks to determine the correctness of a message m containing the update transaction $t_{c,j}$:

- No proof of fraud for transaction $t_{c,j}$ has been published.
- The update transaction $t_{c,j}$ was generated according to underlying channel protocol.
- $j > i$.
- $1 \leq id \leq N$.
- No other message in the same level following the same path has the same id.
- The level of the watchtower that propagates the message, determined through the number of signatures, is equal to the number of rounds passed since the settlement transaction was included in the blockchain.

- For every message in the same path as m: the pk of the previous message corresponds to the secret key sk signing the current message.

If any of the points above are violated the network will reject m.

Payoff for Miners.

The miner including the settlement transaction is paid as in any other transaction. The miner including a valid proof of fraud is paid by the watchtower whose message he included. The reward for the watchtower should thus be high enough to cover the transaction fees.

2.2 Incentive & Security Analysis

In this section we show that watchtowers maximize their profit if they follow the DCWC protocol. Hence, executing the protocol is a dominant strategy and thus the protocol is incentive compatible.

2.2.1 Expected Payoff

In order to show incentive compatibility we need to calculate the expected payoff and analyze changes in the expected payoff when a node decides to deviate from the protocol. The expected payoff of watchtower W watching channel c is

$$\begin{aligned} E[pay_{W,c}] &= \textstyle\sum_{i=1}^{l} P[S_{W,i}] \cdot \frac{\rho_c}{i} \\ &= \textstyle\sum_{i=1}^{l} \sum_{m \in S_{W,i}} P[m] \cdot \frac{\rho_c}{i} \\ &= \textstyle\sum_{i=1}^{l} |S_{W,i}| \cdot P[m] \cdot \frac{\rho_c}{i} \end{aligned} \tag{2.2}$$

where $S_{W,i}$ is the set of messages signed by W such that $d_m = i$, including those messages forwarded by W. Then $P[S_{W,i}]$ denotes the probability that one of those messages will be included in the blockchain. The equally shared payoff ρ_c is split between all watchtowers that forwarded m, paid if m gets included on the blockchain. $P[m]$ is the probability that update message m will eventually be included in the blockchain. To calculate this probability for a message sent in the Disclose step of Phase 1, we sum the probabilities of m being chosen for all possible subsets of non-failing watchtowers. For messages generated in during Cascade to be included in the blockchain, all watchtowers holding messages that travelled fewer hops must have failed. If we assume all nodes follow the protocol and nodes fail independently with probability α, then the probability that message m will be included in the blockchain is:

$$P[m] = \left\{ \begin{array}{ll} \sum_{i=1}^{N} \frac{1}{N}(1-\alpha)^i(\alpha)^{N-i}\binom{N}{i}, & \text{for } d_m = 1 \\ \sum_{i=1}^{|S_i|}(\frac{1}{|S_i|}(1-\alpha)^i(\alpha)^{|S_i|-i}\binom{|S_i|}{i}) \cdot \alpha^{\sum_{j=1}^{i-1}|S_j|}, & \text{for } 1 < d_m \leq l \\ 0, & \text{for } d_m > l \end{array} \right\} \tag{2.3}$$

Where $|S_i|$ is the total number of valid send messages with depth i. If the protocol is executed faithfully by all parties, then $|S_i| = N^i$.

Lemma 2.1. *A watchtower cannot increase his expected payoff $E[pay_{W,c}]$ by deviating from the protocol unless he can decrease d_m for any of his messages, increase the probability $P[m]$ that one of his messages will be included by decreasing the number $|S_i/S_{W,i}|$ of valid messages not signed by him or increase the number $|S_{W,i}|$ of messages signed by him for any i.*

Proof. Lemma 2.1 follows directly from equations 2.2 and 2.3. □

Lemmas 2.2, 2.3, 2.4 complete our argument by showing that no watchtower can decrease d_m or increase $P[m_W]$ or $P[S_W \setminus m_W]$. We prove this for Phase 1 and Phase 2 of the protocol independently.

2.2.2 Analysis of Phase 1: Generating Messages

Lemma 2.2. *A watchtower W can not decrease $|S_i/S_{W,i}|$ nor increase $|S_{W,i}|$ by deviating from executing Phase 1 of DCWC.*

Proof. Picture the Disclose & Cascade algorithm as a tree. W can increase $E[pay_{W,c}]$ by increasing the number of valid messages with his signature. This is true because, whenever a message m from a node in his subtree is published $pay_{W,c} > 0$ and in all other cases $pay_{W,c} = 0$. Note that maximizing the number of messages $|S_{W,i}|$ in any layer bears no effect on the probability of messages of earlier levels to be included; thus W doesn't decrease his chances of a larger payoff by maximizing the number of valid messages in any level of his subtree. Therefor we can analyze each level individually.

It still remains to be shown that W can neither decrease $|S_i/S_{W,i}|$ nor increase $|S_{W,i}|$. Let M denote the set of all update messages. The protocol could only be exploited in one of two ways:

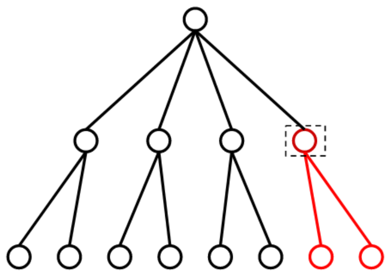

Figure 2.3: Vizualization of Disclose & Cascade Algorithm as a tree, where W is the watchtower in the dotted box.

1. Decrease $|S_i/S_{W,i}|$: Decreasing the number of update messages that are not in his subtree would increase the chance that one of Ws messages gets included.

W has no knowledge of the identities of watchtowers that are not in his subtree. Additionally their success does not depend on any action of W. Thus, decreasing $|S_i/S_{W,i}|$ is not feasible for W.

2. Increase $|S_{W,i}|$: We show that W can't increase the number of nodes in his subtree.

For each depth the number of messages that can be send is restricted by N. If W tries to forge more messages, then according to the pigeonhole principle either two *ids* must be the same, or at least one *id* must be larger than N. In the second case, all descendant messages will be rejected. The first case is a bit trickier. Sending two different messages with the same sequence number to disjoint set of miners is clearly not more beneficial than just sending one message to all. However if the failure probability of nodes is high, creating two messages with the same *id* increases the chances of one of them to get through. This is true when:

$$\begin{aligned} 2 \cdot \alpha \cdot (1-\alpha) &> (1-\alpha)^2 \\ \implies \alpha &> \tfrac{1}{3} \end{aligned} \tag{2.4}$$

However this behavior doesn't scale arbitrarily and whenever $\alpha > \frac{1}{3}$ this behavior increases the security of the protocol.

□

Lemma 2.3. *A watchtower W can not Decrease $d_m | m \in S_{W,i}$ $\forall i$ by deviating from executing Phase 1 of DCWC.*

Proof. In Phase 1, W is only required to store the message m that is last signed by him at this point. The only way to deviate from the protocol would be to forge a message $\tilde{m}$ s.t. $d_m > d_{\tilde{m}}$. We show that W won't be able to construct such a message. If W doesn't control any nodes in the tree which have a shorter path to the root, then he can not create a valid $\tilde{m}$, due to the layered construction of update messages. If W does control a watchtower in the tree which is closer to the root, receiving messages after i hops. Then m is in $S_{W,i}$ which means that we can apply lemma 2.2, as creating $\tilde{m}$ would be equivalent to creating more than N messages at level i. $\square$

2.2.3 Analysis of Phase 2: Committing Messages

The second phase of the protocol handles the spreading of a valid proof-of-fraud through a update message m. Watchtowers might want to withhold proof of frauds, publish them early or deviate in other ways to increase their expected payoff.

Note that there is no need to show that W is not able to decrease d_m in this phase, as we assume that Phase 1 has been completed and all update messages have been created accordingly. It is also easy to see that W can't decrease $|S_i/S_{W,i}|$ as we assume that Phase 1 has already completed and W has no control over the choices that the owners of $|S_i/S_{W,i}|$ take.

Lemma 2.4. *A watchtower W can not increase $|S_{W,i}|$ by deviating from executing Phase 2 of DCWC.*

Proof. W can take three different actions to try increasing the probability that his message will be included in the next block. We show that none of them in fact increase that probability. Firstly, W could send $m \in S_{W,i}$ too early, e.g. in round $j < i$. In this case the transaction is invalid, the network will reject the message and might disconnect from W. Secondly, W could send m in a later round. It is easy to see that there is no advantage in doing so. Lastly, W could send $\tilde{m}$ which belongs to an older update transaction than m_W but a newer one than settlement transaction. If $\tilde{m}$ is in fact included in the blockchain, then the protocol terminates successfully nevertheless. However, there is no incentive for W to store $\tilde{m}$ as storing m instead is a dominant strategy, since the set of potential settlement transactions for which m is a valid proof-of-fraud is a subset of the set of settlement transactions for which $\tilde{m}$ is a valid proof-of-fraud. $\square$

Theorem 2.5. *DCWC is incentive compatible.*

Proof. Theorem 2.5 follows directly from Lemmas 2.1, 2.2, 2.3, 2.4. $\square$

Mining Nodes

Mining nodes might also deviate from the protocol to improve their payoffs. Their set of actions is a bit more limited as any block containing invalid update messages would be rejected by other nodes. The analysis thus far has assumed that the set of watchtowers is disjoint from the set of mining nodes. A node W which is mining a new block and is holding a valid update message, can give himself an advantage of receiving the payoff by not randomly including a proof-of-fraud but rather including his own. This behavior does not influence the outcome of the protocol as it leads to a proof-of-fraud being published either way. The mining node can only include a valid proof-of-fraud in his block without invalidating the block. Thus mining nodes have an additional incentive to participate as watchtowers in our protocol.

2.3 Adaption for Compatibility with Lightning: The DCWC* Protocol

So far, we described the protocol to incentivize watchtowers to watch the channels in high level. An implementation of the protocol depends on the underlying channel protocol and the underlying blockchain protocol. An implementation in Turing complete blockchains can be done as is. An implementation for blockchain protocols that support more limited scripting languages, such as Bitcoins' Script, would require changes to the protocol. Most prominently limiting the number of forwarded messages at each step is, to the best of our knowledge, not possible to be implemented for Lightning due to the limitations of Script. Furthermore, determining how to pay the watchtowers without explicitly naming them in the funding or update messages is not trivial. The protocol depends not only on the scripting language, but also on the channel implementation. We propose a simplified version of our protocol that leaves unspent transaction outputs to be claimed by watchtowers, designed for the current Lightning implementation [4]. DCWC* can be implemented without requiring changes to the channel protocol.

Just as in DCWC, DCWC* creates layers of watchtowers. The i-th layer of watchtowers is allowed to issue a proof of fraud after i rounds trail the occurrence of a fraudulent settlement transaction, also similar to the DCWC protocol. The protocol terminates after a predetermined number of blocks trail the settlement transaction, making funds and rewards spendable. Rewards are granted for the watchtower that submitted the published proof of fraud to the miners and the watchtowers that forwarded that proof of fraud.

The first phase, DISCLOSE & CASCADE*, is executed once for every off-chain update of the channel. This phase is responsible for spreading the update in the network of watchtowers.

2.3.1 Phase 1: Disclose & Cascade*

This phase initiates after the channel is opened and ends when a settlement transaction has been published in the blockchain. SPREAD ensures that a certain amount of watchtowers receive update messages, and that the messages are constructed such that watchtowers receive a payoff for storing and forwarding the messages.

Whenever B receives an update transaction from A, he also receives a transaction that invalidates the previous transaction, rewarding B with all of the channels funds, if the previous state is published on-chain by A. We refer to this transaction, invalidating update transaction $t_{c,i}$ as *invalidation transaction* $\tilde{t}_{c,i}$, which corresponds to a proof-of-fraud in DCWC. B wants to be certain that some watchtowers are aware of this invalidation transaction. Thus, B sends$\tilde{t}_{c,i}$ to watchtowers, claiming most of the funds for himself but leaving some unspent. When a watchtower W receives such a message, he claims some of the unspent transaction outputs (UTXOs) for himself but leaves some unspent and forwards the message. W timelocks the transaction which he added to the message. This way, he will always get a chance to publish a proof of fraud before whomever he sends it to and thus has no incentive not to forward the update. Note that if W later gets the chance to publish a proof of fraud he can still claim all UTXOs for himself. Only if the recipient of his message publishes the proof, W receives the part he claimed before forwarding the message. If the unclaimed UTXO is too small, the receiver will possible discard the message; thus a market of self regulation evolves to determine the appropriate amount for the watchtowers' fee.

The construction of these update messages is depicted in Figure 2.4. If A and B agree on a new update $t_{c,i}$ of the channel they provide each other with the invalidation transaction $\tilde{t}_{c,i-1}$ of the previous transaction. Then each party can create one transaction, giving himself all of the funds in the channel and another transaction with a transaction-level relative timelock (*nSequence*) to all his neighbours, leaving them in control of parts of the output. This can be repeated recursively by his neighbours until the sum of relative timelocks exceeds the timelock of the channel, or until it is reasonable to assume that adding another level makes the participation of more watchtowers unprofitable.

2.3.2 Phase 2: Watch & Commit*

The second phase of the protocol ensures that the fraud will be detected and proven, if it occurs. This phase initiates after the a settlement transaction $t_{c,i}$ is published on the blockchain by one of the involved parties. Since all $t_{c,i}$ are timelocked, the COMMIT phase consists of a fixed number of rounds. Any watchtower holding $\tilde{t}_{c,i}$ can now send it to the blockchain network. Then they can, round after round, append the transactions leading to their payoff.

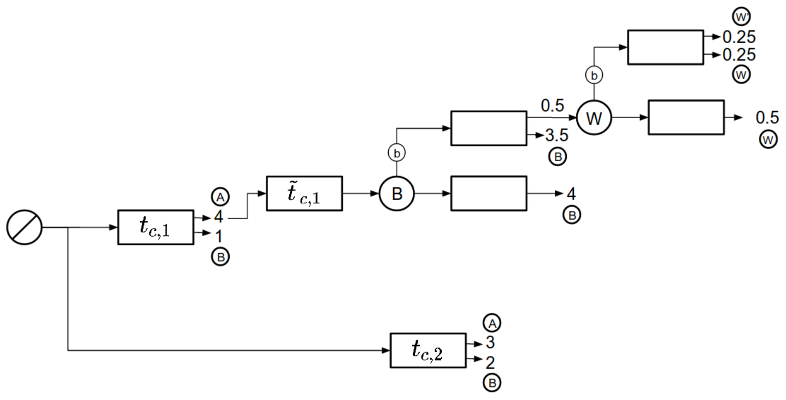

Figure 2.4: Participant B in the channel DISCLOSING the newest invalidation message $\tilde{t}_{c,1}$ to Watchtowers W after A and B agreed on a new update $t_{c,2}$. W CASCADES the message to W' with a transaction level relative timelock b.

```
OP_IF
    b OP_CHECKSEQUENCEVERIFY
    <W's public key>
    OP_CHECKSIG
OP_ELSE
    <B's public key>
    OP_CHECKSIG
```

Figure 2.5: Caption

2.3.3 Implementation

Our protocol has only one type of no-trivial transaction. Namely, the transaction that is forwarded to a watchtower (depicted as a circle in fig. 2.4).It consists of two outputs, one going to the sender of the message B and one that is locked by the script shown in figure 2.5. The forwarded transaction has the purpose of paying the sender of the transaction and the watchtower receiving it, if the sender is offline when the message is included on-chain. The output script is constructed as follows:

Recall that the funding transaction, $t_{c,i}$ and $\tilde{t}_{c,i}$ are part of the channel protocol and not further discussed here.

2.3.4 Discussion of DCWC*

- *Abolishing rounds.* In a practical setting it would make sense to abolish rounds, and with that the CASCADE process all together. This would lead to a higher workload for the participants of the channel as they have to create more messages, but reduces the number of transactions that have to be included on-chain.

- *eltoo.*It remains to be seen how a version of DCWC* would work with eltoo [6] channels. While the DCWC protocol requires watchtowers to store only $O(1)$ messages, DCWC* requires watchtowers to store $O(n)$ messages, where n is the number of channel updates. Possibly the proposed SIGHASH_NOINPUT could be used to improve the implementation of DCWC* to require storing only $O(1)$ messages.

- *privacy.* In DCWC* watchtowers receive a transaction tx, which does not leak more information than the id of the input transaction of tx. Thus, the watchtowers do not learn the distribution of funds.

Chapter 3

Robust Sharded Transaction Ledgers

Sharding protocols are designed to be a scale out solution, thus increasing the numbers of validators and miners by reducing their hardware requirements through splitting responsibilities and not needing every node to see everything. Ideally, in terms of usability and security it should not matter if a distributed transaction ledger is a single blockchain or a sharded system. There are established properties that any robust transaction ledger needs to have, *liveness* and *persistence* under the honest majority assumption. Thus, a sharded protocol should provide similar properties under similar conditions. We give adapted definitions for *liveness* and *persistence*, providing a model for robust sharded transaction ledgers.

In a second step we discuss a high level framework on how to achieve these properties. We split this framework in two parts, first we discuss to achieve liveness and persistence for intra-shard transactions. In a second step we discuss how to achieve liveness and persistence for inter-shard transactions.

3.1 Model & Definitions

In this paper we define properties that robust sharded transaction ledgers should achieve. In order to define these properties we assume that each individual shard achieves *persistence* and *liveness* as defined in [20] under honest majority assumptions for miners of that shard.

Definition 1 (Persistence). Parameterized by $k \in \mathbb{N}$ (the "depth" parameter), if in a certain round an honest player reports a ledger that contains a transaction tx in a block at least k blocks away from the end of the ledger (such transaction will be called "stable"), then whenever tx is reported as stable by any honest player it will be in the same position in the ledger.

Definition 2 (Liveness). Parameterized by $u, k \in \mathbb{N}$ (the "wait time" and "depth" parameters, resp.), provided that a valid transaction is given as input to all honest players continuously for the creation of u consecutive blocks, then all honest parties will report this transaction at least k blocks from the end of the ledger, i.e., all report it as stable.

Definition 3 (Sharded ledger). A sharded ledger is a collection of homogeneous blockchains that support inter-chain transactions between them. The ledger has a shared mining pool, where each miner/validator is at any point in time assumed to have full knowledge over a single shard but only partial knowledge on the other shards (i.e. block headers, merkle proofs of single transactions from other shards).

where an inter-shard transaction consists of at one or more input transactions in one or more shards and one or more output transactions in one or more shards.

3.2 Defining Robust Sharded Transaction Ledgers

We denote the ordered set of *valid* transactions seen by player P working on shard S_i as $\text{OT}\textsc{rans}^l_{P,S_i}$ that are within the prefix of the first l blocks of Ps view on S_i. This includes all transactions included in blockchain S_i as well as all input and output transactions of inter-shard transactions involving S_i. Then let $\textsc{Trans}^l_{P,S_i,S_j}$ denote the unordered set of transactions that player P working on shard S_i sees that are inter-shard transactions involving shard S_i and shard S_j, that are within the prefix of the first l blocks of Ps view on S_i. The notion of a *valid* transaction is of importance as inter-shard transactions could potentially be unrolled, thus become invalid, if this happens it is important to make sure that all related transactions are unrolled as well.

Definition 4 (Persistence in sharded ledgers). Parameterized by $k \in \mathbb{N}$ (the "depth" parameter), for any two honest players P_1 and P_2, working on shards S_i and S_j respectively, then let l be the index of the shorter ledger reported by P_1 or P_2 after truncated by k blocks it holds that:

$$\begin{aligned} &i = j \implies \text{OT}\textsc{rans}^l_{P_1,S_i} = \text{OT}\textsc{rans}^l_{P_2,S_j} \\ &\mathit{and} \\ &i \neq j \implies \textsc{Trans}^l_{P_1,S_i,S_j} = \textsc{Trans}^l_{P_2,S_j,S_i} \end{aligned} \tag{3.1}$$

Note that, while intra-shard transactions eventually have a total order the same does not hold true for inter-shard transactions.

Definition 5 (Liveness in sharded ledgers). Parameterized by $u, k \in \mathbb{N}$ (the "wait time" and "depth" parameters, resp.), provided that any transaction, given as input to all honest players continuously for the creation of u consecutive blocks, then all honest parties will report this transaction at least k blocks from the end of the ledger, i.e., all report it as stable.

Note that from *liveness in sharded ledgers* it follows that each shard needs to have the *liveness* property, as a single inter-shard transaction could have an in- or output in every shard.

Definition 6 (Robust sharded transaction ledger). Protocol Π implements a robust sharded ledger if it has the *persistence in sharded ledgers* and the *liveness in sharded ledgers* properties.

3.3 Designing a Robust Sharded Transaction Ledger

In this section we briefly discuss a framework of intermediate properties that, if achieved, provide a protocol that implements a robust sharded transaction ledger.

3.3.1 Achieving Persistence and Liveness for Intra-Shard Transactions

Definition 7 ((α,β)-Approximate shared decentralization). A sharding protocol has (α,β)-approximate shared decentralization if as long as an adversary controls at most α of mining power in the system, it follows that he can have at most β of mining power in any single shard, where $\alpha \leq \beta$.

If we assume an adversary with α of the total mining power and the individual shards provide *liveness* and *persistence* under an adversary that controls β of the mining power within that shard, then it is enough to provide *(α,β)-approximate shared decentralization* in order to provide *liveness* and *persistence* for intra-shard transactions.

Unfortunately it seems extremely difficult to achieve true *(α,β)-approximate shared decentralization* without having all miners/validators needing to have full knowledge of all shards. The best method to approximate true *(α,β)-approximate shared decentralization* that we are aware of, assignes validators to shards through random sampling and rotates these assignments over time, leading to a slightly weaker property:

Definition 8 ((α,β,x)-Approximate shared decentralization). A sharding protocol has (α,β)-approximate shared decentralization if as long as an adversary controls at most α of mining power in the system, it follows that he is expected have more than β of mining power in any shard for mining at most one out of x blocks, where $\alpha \leq \beta$.

This means that even if the adversary never gains more than α of the total mining power, he can temporarily gain more than β of mining power in a single shard. Thus the severity and frequency of these events, as well as the ability to recover from these events needs to be considered when designing a sharding protocol with this property. This weaker property has been achieved by [17] and [18].

3.3.2 Achieving Persistence and Liveness for Inter-Shard Transactions

A key feature of any sharding protocol is the ability to handle transactions between shards. For these inter-chain transactions we want the same guarantees that we have for intra-chain transactions. Namely, *liveness* and *persistence*.

Definition 9 (Inter-chain transaction commit/abort). An inter-chain transaction tx_A is considered committed at time t if at some $t' \leq t$, tx_A is valid and included in all chains that are involved in the transaction or considered aborted if it is either invalid or not included in all these chains.

Inter-chain transactions introduce new security threats. To identify and secure against such threads we define *inter-chain transaction consistency* as a key property.

Definition 10 (Inter-chain transaction consistency). An inter-chain transaction tx_A is considered consistent at time t if at t, tx_A is either valid and included in all chains that are involved in the transaction or either invalid or not included in all these chains.

Liveness - Atomic Commit

Definition 11 (Atomic commit). Parameterized by $u, k \in \mathbb{N}$ (the "wait time" and "depth" parameters, resp.), a sharded system has atomic commits if all inter-shard transactions are committed or aborted with at least k trailing blocks in each shard, after given as an input to all honest players continuously for the creation of u consecutive blocks.

Note that *atomic commit* for transactions with a single input and a single output follows directly from the liveness property of the individual chains. When multiple inputs and outputs from multiple chains need to be handled, it is required to have the possibility of unrolling an transaction if an input is deemed invalid. This has been achieved by the *Atomix* protocol [18] through introducing *proof-of-acceptance* and *proof-of-rejection*, making any transaction either abortable by presenting a single *proof-of-rejection* or committable by presenting all *proof-of-acceptance*s.

A major security concern with the security of the *atomic commit* is the constant probability of failure that an inter-chain transaction remains inconsistent. An input of an inter-chain transaction could be confirmed in one chain, issuing a valid *proof-of-acceptance*. If the block that confirmed the transaction later becomes stale block, then the *proof-of-acceptance* becomes invalid. But, if the transaction has been committed already it can't be unrolled and will remain inconsistent in the system, leading to the spend value to exist in two chains. Thus, atomic commit provides liveness but not persistence.

Note that to the best of our knowledge, state of the art protocols provide only this property for inter-chain transactions. Having a constant probability of failure per transaction and an ever growing number of transactions will inevitably lead to a system failure.

Persistence - Eventual Consistency

Definition 12 (Eventual Consistency). A multi-chain system has eventual consistency if for all inter-chain transactions it holds that after the transaction has been committed or aborted, the probability that the transaction is inconsistent decreases exponentially with time and converges to 0.

Once we show *eventual consistency* for inter-shard transactions, and *persistence* for intra-shard transactions, we can derive that our sharding protocol has *persistence in sharded ledgers.*

3.4 Achieving Eventual Consistency: The Track & Unroll Protocol

We introduce the TRACK & UNROLL protocol. It provides eventual consistency by tracking dependencies and implicitly unrolling invalid transactions. The protocol we describe is designed handles inter-shard transactions with a single input transaction. Dependencies are in this case the block id's of blocks that include input transactions that the transaction depends on. It can be checked whether the block still exists in the longest chain, by just downloading the block headers. We first describe a protocol for the case of two chains and then expand it to handle n chains.

3.4.1 Track & Unroll for two chains:

When only two chains are involved, solving the problem displayed in 2.1 is straight forward. Miners of B are full clients on B and light clients on A. When a new block for B is mined, the miner of that block checks whether the block header of the block including tx_A is still in chain A. If not, all transactions that spent an input originating from tx_B are implicitly declared invalid, and transactions using them as inputs will be rejected.

3.4.2 Track & Unroll for n chains:

If a third chain C receives an inter-chain transaction tx_C from B, which depends on the validity of tx_B, it will inherently depend on tx_A as well. Let us denote

tx'_B the initiating transaction of tx_C. Miners of C should not be expected to be full clients on B and thus they do not know about the dependency of tx_C on tx_A. Therefore, tx'_B is only valid if it includes all dependencies of all inputs used by tx'_B. Since tx'_B is accessible to miners of C, they can track all dependencies.

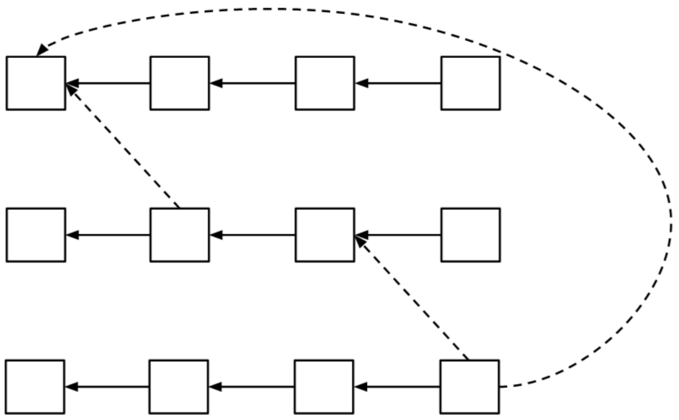
Figure 3.1: Tracking dependencies across multiple chains.

3.4.3 Transaction validation

All transactions are only valid if they meet the conditions of the underlying protocol for that transaction. Additionally all dependencies (block ids) of transactions that were given as an input must still be in their respective longest chain. If any check fails, the transaction is invalid. For intra-shard transactions no further checks are needed.

- Inter-shard input transaction: check if the input transaction explicitly and correctly lists all of his dependencies.
- Inter-shard output transaction: check existence of input transaction. Add the block headers id of that block to the dependency list, or if there was an earlier dependency within the same chain, update that dependency.

3.4.4 Overhead Discussion

A concern of the proposed protocol could be that the overhead of tracking dependencies outweighs the performance gains of using a sharding protocol. So, let us take a look at the actual overhead.

Stale transaction rate:

Implicitly unrolling all subsequent transactions could potentially lead to many transactions being unrolled and lead to longer confirmation times. However, stale blocks becomes highly unlikely when the underline protocol uses byzantine agreement on newly issued blocks. Additionally, one could introduce a waiting time before accepting inter-chain transactions to further reduce this rate.

Transaction size:

Explicitly listing all dependencies increases the transaction size. Note that storing these dependencies on-chain is only necessary for inter-chain transactions.The number of dependencies any transaction needs to store is upper bounded by $n-1$, since we only need to store one dependency per chain. Those dependencies would either conflict, making the transaction invalid or be redundant as an older block on the same chain is always included if a newer one is included. In a practical setting, the size of these dependency list could further drastically be shrunken if we do not track dependencies that are 'stable' or more than k blocks deep.

3.5 Discussion: Designing Secure Cross-Chain Protocols

Sharding and cross-chain protocols share many similarities but are fundamentally different. Sharding aims to design a single, high performing ledger through multiple, *homogeneous* blockchains. Therefor sharding protocols typically assume a shared pool of all miners, following the same protocol. Cross-chains aim at enabling secure interaction between multiple *heterogeneous* blockchains. Each blockchain having their own pool of miners. If poorly designed, the security of such a system would rely on the security of the weakest link - the security of the blockchain with the lowest mining power invested in it. This is why besides *liveness* and *persistence*, cross-shard protocols should have an additional property: *isolated value domains*. This property prevents double spending between chains, even if single chains have a dishonest majority.

Definition 13 (Isolated Value Domains). A multi-chain system is trustless if an adversary gaining full control of one or multiple chains can not gain more financial advantage then the sum of value controlled by these chains.

Isolated value domains add a strong security guarantee. Namely it protects chains from other chains that are controlled by an malicious majority over extended periods of time. It can be interpreted as avoiding double spending attacks between chains. A formal model for robust cross-chain protocols remains to be defined.

Chapter 4

Related Work

For our achievement of securing payment channels through incentivizing watchtowers, the closest work related to ours is the Pisa protocol [21]. Similarly to our paper, [21] introduces and pays third parties to watch channels. There are several differences between their and our approach, making each protocol suitable for different use cases. In [21] a payment channel with each watchtower is set up and each watchtower is obligated to deposit security funds, while our approach pays watchtowers only in case of success and without requiring security deposits, allowing anyone running a full node to become a watchtower without liquidity requirements. Thus our approach compares more to a open market, in which a single honest or rational party is assumed to exist in a large group of witnesses, while [21] introduces a service contract between two (or possibly more) parties. Furthermore we provide the protocol DCWC* which is implementable on top of the lightning network protocol [4], whereas [21] does not seem to be compatible with lightning [4].

For our achievements in defining a secure sharded ledger, the closest related work is the analysis of the Bitcoin backbone protocol [20] and specifically the definition of a robust transaction ledger. The definition of a robust transaction ledger models ledgers that build on a single blockchain and thus does not translate directly to sharded protocols. We achieved this adaption of the model. The Omniledger protocol [18] is one of the only formal protocols that implements a sharded ledger. Our solution for eventual consistency is compatible with the Omniledger protocol and could be adapted easily to improve the protocol and remove an expected time of failure.

Chapter 5

Conclusion & Future Work

In this work we defined a model of properties that a sharding protocol should provide. We provide a framework for designing such a protocol and a high level approach on how to achieve eventual consistency. A meaningful follow-up work would provide a protocol that implements a robust sharded transaction ledger and prove that it is indeed one according to our model. More work could be done on expanding the track & unroll protocol for multiple input transactions from multiple shards. Furthermore, a model similar to ours could be defined for cross-chain, extending and formalizing our discussion on cross-chain protocols.

Furthermore we presented a novel solution to secure channels through incentivizing watchtowers. For payment channels of smaller volume the protocol lets users participate in a channel network without running a full node, thus making payment channels accessible to everybody. The protocol is incentive compatible and has negligible cost in communication and storage for participants.

We suggest building a trust network of nodes that watch each others contract on top of our protocol. Note that it is not specified on which network topology the protocol runs on or how neighbors are found. A simple and reasonable choice would be to run the protocol on top of the P2P network used by the blockchain protocol.

For future work, the protocol could be implemented for the lightning network. For the Raiden network or other turing-complete protocols DCWC could be implemented.

Bibliography

[1] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger via sharding," 2017.

[2] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security.* ACM, 2016, pp. 17–30.

[3] "Ethereum white paper." [Online]. Available: https://github.com/ethereum/wiki/wiki/White-Paper

[4] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," 2015. [Online]. Available: https://lightning.network

[5] "Bitcoin contracts," https://en.bitcoin.it/wiki/Contract, accessed: 2018-09-18.

[6] C. Decker, R. Russell, and O. Osuntokun, "eltoo: A simple layer2 protocol for bitcoin."

[7] C. Decker and R. Wattenhofer, "A fast and scalable payment network with bitcoin duplex micropayment channels," in *Stabilization, Safety, and Security of Distributed Systems*, A. Pelc and A. A. Schwarzmann, Eds. Cham: Springer International Publishing, 2015, pp. 3–18.

[8] G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, and S. Ravi, "Concurrency and privacy with payment-channel networks," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security.* ACM, 2017, pp. 455–471.

[9] P. McCorry, M. Möser, S. F. Shahandasti, and F. Hao, "Towards bitcoin payment networks," in *Information Security and Privacy*, J. K. Liu and R. Steinfeld, Eds. Cham: Springer International Publishing, 2016, pp. 57–76.

[10] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[11] "Raiden network," 2017. [Online]. Available: http://raiden.network/

[12] S. Dziembowski, L. Eckey, S. Faust, and D. Malinowski, "Perun: Virtual payment hubs over cryptographic currencies," IACR Cryptology ePrint Archive 2017, Tech. Rep., 2017.

[13] "Using lightning," https://medium.com/andreas-tries-blockchain/bitcoin-lightning-network-2-we-must-first-become-the-lightning-network-49c46953c1d7, accessed: 2018-07-27.

[14] "Watchtowers," https://www.coindesk.com/laolu-building-watchtower-fight-bitcoin-lightning-fraud, accessed: 2018-06-29.

[15] G. Wood, "Polkadot: Vision for a heterogeneous multi-chain framework," *White Paper*, 2016.

[16] J. Poon and V. Buterin, "Plasma: Scalable autonomous smart contracts," *White paper*, 2017.

[17] "[ETH]Sharding Q&A," https://github.com/ethereum/wiki/wiki/Sharding-FAQs, accessed: 2018-06-19.

[18] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger." *IACR Cryptology ePrint Archive*, vol. 2017, p. 406, 2017.

[19] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 3–16.

[20] J. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2015, pp. 281–310.

[21] P. McCorry, S. Bakshi, I. Bentov, A. Miller, and S. Meiklejohn, "Pisa: Arbitration outsourcing for state channels." *IACR Cryptology ePrin-https://v2.overleaf.com/project/5ba778569f7eee4d151d5178t Archive*, vol. 2018, p. 582, 2018.