Fundamental Properties of the Layer Below a Payment Channel Network

Matthias Grundmann and Hannes Hartenstein

Institute of Telematics, Karlsruhe Institute of Technology, Karlsruhe, Germany
{matthias.grundmann,hannes.hartenstein}@kit.edu

Abstract. Payment channel networks are a highly discussed approach for improving scalability of cryptocurrencies such as Bitcoin. As they allow processing transactions off-chain, payment channel networks are referred to as second layer technology, while the blockchain is the first layer. We uncouple payment channel networks from blockchains and look at them as first-class citizens. This brings up the question what model payment channel networks require as first layer. In response, we formalize a model (called RFL model) for a first layer below a payment channel network. While transactions are globally made available by a blockchain, the RFL model only provides the reduced property that a transaction is delivered to the users being affected by a transaction. We show that the reduced model's properties still suffice to implement payment channels. By showing that the RFL model can not only be instantiated by the Bitcoin blockchain but also by trusted third parties like banks, we show that the reduction widens the design space for the first layer. Further, we show that the stronger property provided by blockchains allows for optimizations that can be used to reduce the time for locking collateral during payments over multiple hops in a payment channel network.

1 Introduction

Payment channel networks (PCNs) became popular as an approach for improving the scalability of blockchain based cryptocurrencies such as Bitcoin [12]. While Bitcoin scales well in the amount of coins that can be transferred by a transaction, it can only process a limited number of transactions per second. PCNs, e.g., the Lightning Network [14] for Bitcoin, perform transactions off-chain in a second layer and they do not require global consensus for every transaction as long as all participants are honest.

PCNs have mostly been looked at as second layer on top of a blockchain. While the idea to use banks as a first layer has already been mentioned in 2015 by Tremback and Hess [17], to the best of our knowledge it has not been analyzed which properties PCNs require for the layer below. We look at PCNs as first-class citizen independent from the specific first layer and analyze whether PCNs can be used on top of models that are different from a blockchain (with the term blockchain, we refer to a public permissionless blockchain such as Bitcoin). As an affirmative answer, we present a reduced **model for the first layer** which

we call **RFL model (Reduced First Layer)** whose properties give reduced guarantees compared to a blockchain but yet suffice to implement a protocol for a PCN on top. The reduced property of the RFL model in comparison to a blockchain is that a blockchain delivers each transaction to all peers participating in the network while in the RFL model a transaction is only guaranteed to be visible for a transaction's affected users. A transaction's affected users are the users who receive the transferred coins and the users who were able to spend the same coins that the transaction transfers. To show that a PCN can still securely be implemented using the RFL model as underlying first layer, we show the following property: A payment channel between two users can be closed within a given time so that each user u receives at least their correct balance if u is honest and checks the first layer regularly for new transactions. We will refer to this property as the security property and formally define it. We **prove this security property** for a PCN protocol that is executed on a first layer that implements the properties defined by the RFL model: liveness, affected user synchrony, persistence, and transaction validity.

Our analysis of the relationship between the RFL model and a blockchain shows that a blockchain instantiates the RFL model under typical assumptions for blockchains such as Bitcoin. Having shown that a PCN can be implemented on a reduced model compared to a blockchain, we show that the RFL model can not only be instantiated by blockchains but there is a wider design space for the first layer, e.g., using trusted third parties like banks. Implementation of PCNs on different first layers allows for a range of design opportunities, e.g., with respect to trust, privacy, liquidity, online requirements, and currencies. Having seen the advantages of a model for the first layer that is reduced compared to a blockchain, we also look at the advantages of using a blockchain that guarantees more than the RFL model: A blockchain's property to deliver a transaction to everyone can be used to **optimize payments in PCNs** so that collateral is locked for a shorter amount of time [11]. We show that this requires stronger assumptions than the basic construction of PCNs.

For readers, being not familiar with payment channels and PCNs, we provide a section on fundamentals in the extended version of this paper [6]. In the following section, we put our work in the context of related work. In Section 3, we present the RFL model. We show in Section 4 that a simplified version of the Lightning Network's protcol fulfills the security property if it is used on a first layer that instantiates the RFL model. Section 5 presents and compares instances of the RFL model using a blockchain and trusted banks. In Section 6, we show how the stronger properties fulfilled by a blockchain can be used to optimize payments in a PCN. Finally, we conclude in Section 7.

2 Related Work

Kiayias and Litos provide in [10] a formalization and security analysis of the Lightning Network using a global ledger functionality modeled in [2]. The global ledger fulfills a global synchrony property: An honest user that is connected to

the required resources is being synchronized (receives the latest state) within a bounded time. Their work is orthogonal to our work because it appears that their proof would still work with the assumptions of the RFL model. We leave it for future work to provide a security proof for the Lightning Network that uses the RFL model instead of the global ledger functionality provided by Bitcoin.

Credit networks as proposed in [3] are a concept that is related to that of PCNs. A credit network, however, does not have an underlying layer. In a credit network, users are connected through credit links (IOUs) which represent the amount one user owes another user. This construction requires users to trust each other to an extent that is quantified by the size of the credit link. In a PCN, users are instead required to trust the first layer and not each other.

Recently, Avarikioti et al. proposed Cerberus channels, [1] a protocol for payment channels that includes watchtowers watching for outdated commitment transactions on the first layer. They also define a security property and show that Cerberus channels fulfill this property. The security property we define is inspired by the security property used in [1]; however, our definition does explicitly consider the timeouts and includes HTLCs which are required for payments over intermediaries.

3 RFL Model of First Layer

In this section, we present the RFL model, a model for the first layer that guarantees a reduced set of properties compared to a blockchain. The main difference of the RFL model to a blockchain is that, when using a blockchain, a transaction is delivered to all users. In this section, we define the *affected user synchrony* property which only requires the first layer to deliver a transaction to the users being affected by a transaction.

In the RFL model, we have a set of users U who can create transactions. Each user $u \in U$ has an asymmetric key pair with private key s_u and the public key p_u. A transaction t consists of an amount of coins and is associated with a set of *receivers* $\Omega_t^{\mathrm{R}} \subseteq U$ that the coins are transferred to. The set of users who were able to spend the coins that are spent by t is referred to as the *potential senders* $\Omega_t^{\mathrm{S}} \subseteq U$. We denote as *affected users* of the transaction t the set $\Omega_t = \Omega_t^{\mathrm{S}} \cup \Omega_t^{\mathrm{R}}$. For a transaction to be valid, it needs to fulfill conditions depending on the coins that are spent (e.g., a signature using a specified key or some timeout having passed). In general, it is possible that not all potential senders need to sign a transaction and thus some potential senders and receivers might not have seen the transaction before it has been published on the first layer.

We model the first layer as a (logically) single party $\mathcal{L}$ that is connected to all other parties via secure and reliable channels. Users can send transactions t to the first layer $\mathcal{L}$. We refer to this action as *publishing* t. $\mathcal{L}$ can send a confirmation that the transaction t has been executed, i.e. the coins have been transferred, and users can query $\mathcal{L}$ whether a transaction has been confirmed. The time a user has to wait for the confirmation is determined by the liveness property parametrized below by the waiting time Δl_{conf}. The first layer $\mathcal{L}$ can send transactions from

other users and confirmations to users. Users can check the first layer $\mathcal{L}$ for the confirmation of a transaction. Whether the first layer's response about the confirmation is consistent among different users, is determined by the affected user synchrony property below parametrized by the time Δl_{sync}.

The first layer $\mathcal{L}$ has to have these essential properties:

- *Liveness*: If a user sends a valid transaction t to the first layer $\mathcal{L}$, then $\mathcal{L}$ will confirm the transaction after at most Δl_{conf}.
- *Affected User Synchrony*: If a user has received a confirmation from $\mathcal{L}$ for a transaction t with affected users Ω_t, then $\mathcal{L}$ makes t and the confirmation visible to all affected users $u \in \Omega_t$ within at most Δl_{sync}.
- *Persistence*: If a user has received a confirmation for transaction t from $\mathcal{L}$, then $\mathcal{L}$ will always report t as confirmed.
- *Transaction Validity*: A transaction t will only be confirmed by $\mathcal{L}$ if t is valid.

Note that instead of the affected user synchrony, a blockchain implements an unrestricted synchrony property: If a transaction is confirmed by the blockchain, the transaction will be seen as confirmed by all users within a given time. We will look deeper into the relationship between the RFL model and blockchains in Section 5.

For the RFL model, we use a UTXO (unspent transaction output) model for transactions which is also used by Bitcoin. In the UTXO model, a transaction consists of multiple inputs and multiple outputs. Each input spends an output of a previous transaction. An output specifies a condition that an input needs to meet to spend the output. An example for a condition is the signature of a public key that is specified in the output and the signature needs to be provided in the input. If a transaction is processed by the first layer, the first layer checks whether the transaction is *valid*. For a transaction to be valid, all inputs have to spend transaction outputs that are unspent and the conditions of the UTXOs need to be met. For a PCN as defined later, we require the following types of conditions: signature corresponding to a given public key, preimage for a given image of a hash function, time since confirmation of UTXO spent, and combinations thereof using logical operators OR and AND. The set of receivers Ω_t^{R} contains all users whose signature is required by at least one condition to spend an output of t. Analogously, the set of potential senders Ω_t^{S} of a transaction t is comprised of all users whose signature is required by at least one condition to spend an output that is spent by t.

4 Security Property for a Payment Channel Network Protocol based on the RFL model

In this section, we show that the RFL model suffices as a first layer to securely implement a PCN. To this end, we make use of a slightly simplified version of the Lightning Network's protocol, define a security property, and then show that the protocol fulfills this security property when it uses the RFL model as first layer. A more detailed description of the protocol can be found in the extended

version of our paper [6]. Here, we only introduce the notation and explain some differences that we assume in comparison to the Lightning Network's protocol specification on Github[1] as a basis for the proof.

The current specification of the Lightning Network's protocol is only for single-funded channels, i.e. a channel is created by putting funds of only one participant in the channel. Payments are executed using HTLCs even if they are direct payments over just one channel. We use Alice and Bob as names for two users participating in the protocol. Alice has a secret key s_{A} associated with the public key p_{A} that she uses for commitment transactions and HTLCs. This is a simplification; the Lightning Network specification uses different keys to sign HTLCs and commitment transactions to allow for separating keys in cold and hot storage[2]. We count the states of a channel using $n \geq 1$ and denote the commitment transaction held by Alice for state n as $t_{\mathrm{C}n\mathrm{A}}$ ($t_{\mathrm{C}n\mathrm{B}}$ for Bob). For revocation, Alice creates a new revocation key $s_{\mathrm{R}n\mathrm{A}}$ for each commitment transaction $t_{\mathrm{C}n\mathrm{A}}$. Each output of Alice's commitment transaction $t_{\mathrm{C}n\mathrm{A}}$ is spendable using the revocation key $s_{\mathrm{R}n\mathrm{A}}$ and Bob's public key, i.e. Bob can spend all outputs of $t_{\mathrm{C}n\mathrm{A}}$ if Alice publishes $t_{\mathrm{C}n\mathrm{A}}$ after she has revoked $t_{\mathrm{C}n\mathrm{A}}$ by sending $s_{\mathrm{R}n\mathrm{A}}$ to Bob. This is another simplification. The Lightning Network uses a construction that generates a revocation key for $t_{\mathrm{C}n\mathrm{A}}$ from a long term secret created by Bob and the per-commitment secrets created by Alice and, to revoke a transaction, Alice shares the corresponding per-commitment secret with Bob.

Each commitment transaction $t_{\mathrm{C}n\mathrm{A}}$ held by Alice for state n of the channel between Alice and Bob is built in the following way ($t_{\mathrm{C}n\mathrm{B}}$ held by Bob is constructed analogously): The commitment transaction's input spends the channel's funding transaction's output which requires Alice's and Bob's signatures. We use the term *stable balance* for a user's balance that is not part of an HTLC. The HTLC outputs of $t_{\mathrm{C}n\mathrm{A}}$ cannot directly be spent by Alice but only using the HTLC timeout transaction $t_{\mathrm{T}ny\mathrm{A}}$ and the HTLC success transaction $t_{\mathrm{S}ny\mathrm{A}}$ which will be explained below. Outputs spendable by Alice are locked for Δt_{comm} time to give Bob time to spend the output in case Alice publishes $t_{\mathrm{C}n\mathrm{A}}$ after it has been outdated. Alice's commitment transaction $t_{\mathrm{C}n\mathrm{A}}$ has the following outputs:

- An output for Alice's stable balance that is spendable
  - by Bob using Alice's revocation key $s_{\mathrm{R}n\mathrm{A}}$ for state n or
  - by Alice after delay Δt_{comm}; aka 'to self delay'.
- An output for Bob's stable balance that is spendable by Bob.
- For each outgoing HTLC an output for the HTLC's balance that is spendable
  - by Bob if he provides a preimage for a given y, or
  - by Bob using Alice's revocation key for state n, or
  - by the HTLC-timeout transaction $t_{\mathrm{T}ny\mathrm{A}}$ after point in time T_y^{HTLC}.
- For each incoming HTLC an output for the HTLC's balance that is spendable
  - by Bob after point in time T_y^{HTLC}, or
  - by Bob using Alice's revocation key for state n, or
  - by the HTLC-success transaction $t_{\mathrm{S}ny\mathrm{A}}$ using preimage for a given y.

[1] https://github.com/lightningnetwork/lightning-rfc/blob/master/00-introduction.md
[2] https://github.com/lightningnetwork/lightning-rfc/blob/master/02-peer-protocol.md#rationale

There are two HTLC transactions per HTLC with preimage y that depend on $t_{\mathrm{C}n\mathrm{A}}$: An HTLC timeout transaction $t_{\mathrm{T}ny\mathrm{A}}$ and an HTLC success transaction $t_{\mathrm{S}ny\mathrm{A}}$. The inputs for the HTLC transactions are the outputs indicated above for the commitment transaction $t_{\mathrm{C}n\mathrm{A}}$ and need to be signed by Alice and Bob. Both HTLC transactions held by Alice have one output that is spendable

- by Alice after delay Δt_{comm}, or
- by Bob using Alice's revocation key for state n.

We define the forwarding timeout delta Δ_{forw} as a global value. This simplifies the Lightning Network's specification which uses values that are chosen by the users of each channel. Lastly, the closing of channels can be cooperative in the Lightning Network specification to allow both parties to access their funds immediately. However, we assume that users simply send their latest commitment transaction to the first layer.

Security Property for the Payment Channel Network Protocol based on the RFL model We now show that the protocol described above fulfills the security property as defined in Lemma 1 when used with a first layer that implements the RFL model. We start by defining the term *correct balance*:

Definition 1. *The **correct balance** of a user Alice is the sum of Alice's stable balance and the amounts of outgoing HTLCs that she has not received the secret for and the amounts of incoming HTLCs that she has received the secret for.*

To facilitate understanding of the following lemma we give a brief review of the most important relative and absolute timings that are used: The first layer $\mathcal{L}$ confirms a valid transaction within Δl_{conf}, other users will see a confirmed transaction within Δl_{sync}, a user can spend their own commitment transaction's outputs after Δt_{comm}, and an HTLC with condition y times out at time T_y^{HTLC}.

Lemma 1. *The protocol as defined above fulfills the property **security**: At each point in time T_{now}, Alice can close the payment channel between Alice and Bob so that she has received at least her correct balance at time $T = \max(T_{\mathrm{max}}^{\mathrm{HTLC}}, T_{\mathrm{now}}) + 2 \cdot \Delta l_{\mathrm{conf}} + \Delta t_{\mathrm{comm}}$ if she is honest and checks the first layer $\mathcal{L}$ for transactions at least once every Δ_{user} and $0 < \Delta_{\mathrm{user}} < \Delta t_{\mathrm{comm}} - \Delta l_{\mathrm{sync}} - \Delta l_{\mathrm{conf}}$, where $T_{\mathrm{max}}^{\mathrm{HTLC}}$ denotes the maximal timeout of all HTLCs in the channel.*

Lemma 1 follows from the following arguments. We first look at the case that Alice initiates the closing of the channel and then at the case that Bob closes the channel and Alice did not want to close the channel. Let n be the number of the latest state. To close the channel, Alice sends her latest commitment transaction $t_{\mathrm{C}n\mathrm{A}}$ and the associated HTLC transactions to $\mathcal{L}$ at T_{now}. Alice has Bob's signature for $t_{\mathrm{C}n\mathrm{A}}$ because she receives Bob's signature for the initial commitment transaction t_{C1A} during the opening of the channel ($n = 1$) and during each update of the channel ($n > 1$), Alice receives Bob's signature for the latest commitment transaction $t_{\mathrm{C}n\mathrm{A}}$ and the associated HTLC transactions. If there are no conflicting transactions, $t_{\mathrm{C}n\mathrm{A}}$ will be confirmed within Δl_{conf} according

to the liveness property of $\mathcal{L}$. The funding transaction can only be spent by Alice and Bob and thus a transaction conflicting with $t_{\mathrm{C}n\mathrm{A}}$ can only be sent to $\mathcal{L}$ by Bob until $t_{\mathrm{C}n\mathrm{A}}$ is confirmed by $\mathcal{L}$. At time $T_{\mathrm{now}} + \Delta l_{\mathrm{conf}}$ one commitment transaction will be confirmed – either Alice's transaction or a transaction sent by Bob. Thus, we need to distinguish three scenarios:

Bob does not publish a commitment transaction: The first layer $\mathcal{L}$ will confirm Alice's commitment transaction $t_{\mathrm{C}n\mathrm{A}}$ within Δl_{conf} according to the liveness property. Alice can spend her stable balance after an additional Δt_{comm}. The associated output cannot be spent by Bob because Bob does not have the revocation key $s_{\mathrm{R}n\mathrm{A}}$ for the latest commitment transaction. For the incoming HTLCs in $t_{\mathrm{C}n\mathrm{A}}$ that Alice has the secret for, she publishes the success transactions $t_{\mathrm{S}ny\mathrm{A}}$ together with $t_{\mathrm{C}n\mathrm{A}}$ and the success transactions' outputs can be spent by Alice after Δt_{comm}. Bob cannot spend the HTLC output because the associated time T_y^{HTLC} has not come (else, Alice would have removed the HTLC or timely gone on-chain) and Bob does not have the revocation keys. Thus, Alice can spend her stable balance and her balance of incoming HTLCs at $T_{\mathrm{now}} + \Delta l_{\mathrm{conf}} + \Delta t_{\mathrm{comm}}$. Her transaction to spend these outputs will be confirmed within Δl_{conf}. For the outgoing HTLCs in $t_{\mathrm{C}n\mathrm{A}}$ that Alice does not have the secret for, she can spend the output using the timeout transaction $t_{\mathrm{T}ny\mathrm{A}}$ after T_y^{HTLC}. The timeout transaction is confirmed by $\mathcal{L}$ and its output is spendable after $T_y^{\mathrm{HTLC}} + \Delta l_{\mathrm{conf}} + \Delta t_{\mathrm{comm}}$. If Bob spends the HTLC using the preimage, Alice receives the preimage because of the affected user synchrony property of $\mathcal{L}$ because Alice is a potential sender of the HTLC output, so the HTLC's amount is not taken into account for this channel's correct balance. Thus, Alice has received her correct balance after at most $\max(T_{\mathrm{now}}, T_{\mathrm{max}}^{\mathrm{HTLC}}) + 2 \cdot \Delta l_{\mathrm{conf}} + \Delta t_{\mathrm{comm}} = T$.

Bob has published his latest commitment transaction $t_{\mathrm{C}n\mathrm{B}}$: In this case, the affected user synchrony property of the first layer asserts that Alice can see Bob's commitment transaction $t_{\mathrm{C}n\mathrm{B}}$ because she is a potential sender. Alice can instantly spend her stable balance in the channel once she sees the transaction $t_{\mathrm{C}n\mathrm{B}}$ at time $T_{\mathrm{now}} + \Delta l_{\mathrm{conf}} + \Delta l_{\mathrm{sync}}$. Alice's transaction spending her stable balance will be confirmed within Δl_{conf}. For each outgoing HTLC, Alice can spend the HTLC output after T_y^{HTLC}. If Bob spends the HTLC output using $t_{\mathrm{S}ny\mathrm{B}}$ by providing a preimage for the given y, Alice receives the preimage because of the affected user synchrony property of $\mathcal{L}$. For each incoming HTLC, Alice must publish a transaction to redeem the HTLC if she has the preimage for the given y which takes Δl_{conf} to be confirmed. Thus, Alice has received her correct balance within $\max(T_{\mathrm{now}} + \Delta l_{\mathrm{conf}} + \Delta l_{\mathrm{sync}}, T_{\mathrm{max}}^{\mathrm{HTLC}}) + \Delta l_{\mathrm{conf}} \leq T$.

Bob has published an outdated commitment transaction: For each update to state number $i + 1$, Alice receives Bob's revocation key $s_{\mathrm{R}i\mathrm{B}}$. Say, Bob has published an outdated commitment transaction $t_{\mathrm{C}o\mathrm{B}}$ with $o < n$. Alice can see the transaction after $T_{\mathrm{now}} + \Delta l_{\mathrm{conf}} + \Delta l_{\mathrm{sync}}$. Bob can only spend his stable output of $t_{\mathrm{C}o\mathrm{B}}$ after an additional Δt_{comm}. In case Bob publishes an HTLC success or timeout transaction, Alice also sees Bob's HTLC transaction because of the affected user synchrony property of $\mathcal{L}$ and Bob can only spend its output after Δt_{comm}. Thus, Alice must use her key s_{A} and Bob's revocation

key $s_{\mathrm{R}o\mathrm{B}}$ to create a revocation transaction that spends the whole balance in the channel (output for Alice, output for Bob, and all HTLC outputs). After Δl_{conf} this revocation transaction has been confirmed. Because $\Delta l_{\mathrm{conf}} < \Delta t_{\mathrm{comm}}$ (see Lemma 1), Bob cannot have spent his outputs before Alice. Thus, Alice has received her correct balance within $T_{\mathrm{now}} + 2 \cdot \Delta l_{\mathrm{conf}} + \Delta l_{\mathrm{sync}} \leq T$.

In case Bob closes the channel by sending $t_{\mathrm{C}i\mathrm{B}}, i \leq n$ to $\mathcal{L}$ at T_{now} and Alice did not want to close the channel, too, Alice receives $t_{\mathrm{C}i\mathrm{B}}$ from $\mathcal{L}$ within $T_{\mathrm{recvA}} = T_{\mathrm{now}} + \Delta l_{\mathrm{conf}} + \Delta l_{\mathrm{sync}} + \Delta_{\mathrm{user}}$ because Alice is an affected user of the transaction and the affected user synchrony property of $\mathcal{L}$ asserts that she can see the transaction within Δl_{sync} and Alice checks the first layer $\mathcal{L}$ at least every Δ_{user} for new transactions. Bob's stable output and HTLC transaction outputs cannot be spent by him until $T_{\mathrm{spendB}} = T_{\mathrm{now}} + \Delta l_{\mathrm{conf}} + \Delta t_{\mathrm{comm}}$. If $i < n$, Alice must use her key s_{A} and Bob's revocation key $s_{\mathrm{R}i\mathrm{B}}$ to spend the whole balance in the channel using a revocation transaction when she sees Bob's transaction at time T_{recvA}. This revocation transaction will be confirmed by the first layer after $T_{\mathrm{recvA}} + \Delta l_{\mathrm{conf}}$. Bob cannot have spent his outputs at this time because $T_{\mathrm{recvA}} + \Delta l_{\mathrm{conf}} = T_{\mathrm{now}} + \Delta l_{\mathrm{conf}} + \Delta l_{\mathrm{sync}} + \Delta_{\mathrm{user}} + \Delta l_{\mathrm{conf}} < T_{\mathrm{now}} + \Delta l_{\mathrm{conf}} + \Delta t_{\mathrm{comm}} = T_{\mathrm{spendB}}$ because $\Delta_{\mathrm{user}} < \Delta t_{\mathrm{comm}} - \Delta l_{\mathrm{sync}} - \Delta l_{\mathrm{conf}} \implies \Delta l_{\mathrm{conf}} + \Delta l_{\mathrm{sync}} + \Delta_{\mathrm{user}} < \Delta t_{\mathrm{comm}}$. Thus, Alice has received her correct balance after $T_{\mathrm{now}} + \Delta l_{\mathrm{conf}} + \Delta l_{\mathrm{sync}} + \Delta_{\mathrm{user}} + \Delta l_{\mathrm{conf}} < T_{\mathrm{now}} + \Delta l_{\mathrm{conf}} + \Delta t_{\mathrm{comm}} \leq T$. If $i = n$, Alice reacts analogously to the case above that Bob has published his latest commitment transaction but the times are postponed by Δ_{user} (see [6]).

5 Instances and Options of the RFL model

The RFL model describes an ideal first layer that guarantees the properties required by a PCN. In this section, we show that, under certain assumptions, a blockchain instantiates such a first layer. We also sketch the idea of an instance of a first layer using a bank or a network of banks and provide a comparative exploration of design options.

Using a Blockchain. Garay et al. show in various works (e.g., [4,5]) that the Bitcoin protocol satisfies *consistency* and *liveness* with high probability under the assumption of a bounded-delay network model and an honest majority of computing power [5]. In comparison to our definition in Section 3, their definition of liveness assumes that a transaction is provided to all honest parties. This is implemented in Bitcoin by flooding transactions in the peer-to-peer network. The definition of consistency used in [5] implies our definition of persistence and affected user synchrony. It is even stronger and implies *synchrony* for all honest peers, i.e. the first layer $\mathcal{L}$ makes a transaction t and the confirmation visible to all honest peers. Assuming an honest majority of computing power and using a bounded-delay network model, the results of Garay et al. show that a blockchain similar to Bitcoin instantiates the RFL model with high probability.

Note that for a blockchain, liveness is not guaranteed because the blocksize is limited and there can be times during which the blockchain is congested so that users have to compete for publishing their transactions on the blockchain.

Recent work [9,18,7] has shown attacks against payment channels that attack the liveness of a blockchain, e.g. by bribing miners to censor transactions. These works show the importance of considering the properties of the first layer when building second layer architectures.

Using a Single or Multiple Banks. Having an abstract model of the first layer allows for developing architectures that instantiate a first layer without a blockchain. For example, a network of banks can be used to instantiate a first layer under the assumption of trust in the banks. We assume the common features of banks as described in the following but the bank does not necessarily need to be a classical bank and can also be a payment service provider. A contemporary bank offers to their consumers an interface that implements liveness, transaction validity, and persistence. The usual visibility for a transaction matches the visibility required by affected user synchrony: A bank makes a transaction visible (only) to the potential senders and receivers of transferred funds. A transaction has multiple potential senders if it is sent from a joint account. Using banks as first layer, their customers could perform transactions using a PCN. While this requires trust into the bank to implement the RFL model, the transactions are hidden from the bank which improves privacy because the bank gains less information. So the PCN could be used for decentralized digital cash.

Comparative Exploration of Design Options. We now discuss differences between using a blockchain and trusted banks as different ways to instantiate the RFL model.

Trust While banks have to be fully trusted, trust into a blockchain is more distributed (e.g., honest miners have the largest share of computation power).

Privacy While PCNs do not categorically improve privacy [8,15,16], the privacy properties of the first layer are crucial for privacy in the PCN. While users are identified by pseudonyms on a blockchain, a bank is required to implement methods for customer identification. This reduces privacy for the users because the bank learns about their transactions. However, it allows the bank to implement access control on the transactions and to make a transaction only visible to the affected users of the transaction which, in turn, improves privacy. By facilitating tracing of money laundering, customer identification can be a way to increase chances of mainstream adoption of a digital payment system.

Liquidity Payment channels require users to deposit funds by locking them on the first layer while the channel is open. On a blockchain, users can only use coins in their channel that they own on the blockchain. Using banks to implement a first layer allows for letting users open channels using credit they receive from their bank. This can improve the liquidity within the network because more users are able to forward payments if they have channels with higher capacity.

Online requirement While the first layer needs to provide affected user synchrony to make transactions visible, the corresponding part for the user is to check the first layer regularly for new transactions to react to outdated commitment transactions. With a blockchain as a first layer, this task requires a user to stay connected and analyze new blocks or to outsource this task to a watchtower. With a centralized system of banks as first layer, the bank needs to be

trusted to run a system that provides affected user synchrony. Because the bank knows their customers, they could even contact them to inform about relevant transactions and, thus, include watchtower functionality in the first layer.

Currencies The two different ways for instantiating the RFL model also differ in the type of currencies they can support. While a blockchain can provide a decentralized currency, a system of banks can make traditional currencies managed by central banks available for use in PCNs.

6 Optimization of HTLCs using a Blockchain

The basic construction of PCNs leaves room for optimizations. One issue is that the amount of collateral that needs to be locked for one transaction allows for balance availability attacks [13] which lock the balance so that it cannot be used by honest parties. Recent research [11] has found ways to reduce the required collateral during one transaction over multiple hops assuming a blockchain as first layer that offers (global) synchrony instead of the reduced affected user synchrony. The Sprites protocol [11] uses a (logically) central "preimage manager" that can be read and written to by any party of the PCN. All channel updates on the route of a payment depend on the condition that the secret x has been published before a specific timeout that is the same across all channels. By using it as a "global synchronization gadget", the preimage manager is used to synchronize the timeouts so that all parts of the route timeout at the same time in contrast to increasing timeouts from the receiver's end in the Lightning Network. As all participants have the same view on the blockchain, either all updates that depend on the publication of the secret before a given timeout fail or all are valid. For a first layer instantiated by banks, a bank or another trusted party could implement such a preimage manager. However, this would add a central entity as dependency for the payments and thus, payments would not be decentralized anymore. This shows that, while the RFL model allows for a PCN to be implemented as second layer, the stronger properties fulfilled by a blockchain enable optimizations for HTLCs. The solution of Sprites can improve the PCN protocol because it makes use of the gap between the properties of the RFL model and a blockchain's properties.

7 Conclusion

For a PCN, a first layer can be used that delivers only a reduced set of properties compared to a blockchain. We defined these properties in the RFL model and showed that this model suffices to implement a secure protocol for PCNs. Furthermore, the RFL model can be instantiated by blockchains. We examined how the difference between the properties that blockchains have in comparison to the RFL model can be used to improve payments over HTLCs and we have shown that this difference has already been used by improvements that have been proposed in previous works. We also showed that banks can instantiate the RFL model. Implementing a first layer might be a role banks play in the future.

References

1. Avarikioti, Z., Litos, O.S.T., Wattenhofer, R.: Cerberus Channels: Incentivizing Watchtowers for Bitcoin (Feb 2020)
2. Badertscher, C., Maurer, U., Tschudi, D., Zikas, V.: Bitcoin as a Transaction Ledger: A Composable Treatment. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology – CRYPTO 2017. pp. 324–356. Lecture Notes in Computer Science, Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_11
3. Fugger, R.: Money as IOUs in a Social Trust Network and A Proposal for a Secure, Private, Decentralized Digital Currency Protocol. Tech. rep. (2004)
4. Garay, J., Kiayias, A., Leonardos, N.: The Bitcoin Backbone Protocol: Analysis and Applications. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology - EUROCRYPT 2015. pp. 281–310. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_10
5. Garay, J., Kiayias, A., Leonardos, N.: Full Analysis of Nakamoto Consensus in Bounded-Delay Networks. Tech. Rep. 277 (2020), https://eprint.iacr.org/2020/277
6. Grundmann, M., Hartenstein, H.: Fundamental Properties of the Layer Below a Payment Channel Network (Extended Version). arXiv:2010.08316 [cs] (Oct 2020), http://arxiv.org/abs/2010.08316, arXiv: 2010.08316
7. Harris, J., Zohar, A.: Flood & Loot: A Systemic Attack On The Lightning Network. arXiv:2006.08513 [cs] (Jun 2020), http://arxiv.org/abs/2006.08513
8. Kappos, G., Yousaf, H., Piotrowska, A., Kanjalkar, S., Delgado-Segura, S., Miller, A., Meiklejohn, S.: An Empirical Analysis of Privacy in the Lightning Network. arXiv:2003.12470 [cs] (Mar 2020), http://arxiv.org/abs/2003.12470
9. Khabbazian, M., Nadahalli, T., Wattenhofer, R.: Timelocked Bribes. Tech. Rep. 774 (2020), https://eprint.iacr.org/2020/774
10. Kiayias, A., Litos, O.S.T.: A Composable Security Treatment of the Lightning Network. In: 2020 IEEE 33rd Computer Security Foundations Symposium (CSF). pp. 334–349 (Jun 2020). https://doi.org/10.1109/CSF49147.2020.00031, iSSN: 2374-8303
11. Miller, A., Bentov, I., Kumaresan, R., Cordi, C., McCorry, P.: Sprites and State Channels: Payment Networks that Go Faster than Lightning. arXiv:1702.05812 [cs] (Feb 2017), http://arxiv.org/abs/1702.05812
12. Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System. Tech. rep. (2008)
13. Perez-Sola, C., Ranchal-Pedrosa, A., Herrera-Joancomartı, J., Navarro-Arribas, G., Garcia-Alfaro, J.: LockDown: Balance Availability Attack against Lightning Network Channels p. 18
14. Poon, J., Dryja, T.: The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. Tech. rep. (2016)
15. Rohrer, E., Tschorsch, F.: Counting Down Thunder: Timing Attacks on Privacy in Payment Channel Networks. arXiv:2006.12143 [cs] (Jun 2020), http://arxiv.org/abs/2006.12143
16. Romiti, M., Victor, F., Moreno-Sanchez, P., Haslhofer, B., Maffei, M.: Cross-Layer Deanonymization Methods in the Lightning Protocol. arXiv:2007.00764 [cs] (Jul 2020), http://arxiv.org/abs/2007.00764
17. Tremback, J., Hess, Z.: Universal Payment Channels (Nov 2015)
18. Tsabary, I., Yechieli, M., Eyal, I.: MAD-HTLC: Because HTLC is Crazy-Cheap to Attack. arXiv:2006.12031 [cs] (Jun 2020), http://arxiv.org/abs/2006.12031

The final publication is available at https://doi.org/10.1007/978-3-030-66172-4_-26.