Towards Bitcoin Payment Networks

Patrick McCorry[1(✉)], Malte Möser[2], Siamak F. Shahandasti[1], and Feng Hao[1]

[1] School of Computing Science, Newcastle University, Newcastle upon Tyne, UK
patrick.mccorry@ncl.ac.uk, siamak.shahandashti@newcastle.ac.uk,
feng.hao@newcastle.ac.uk
[2] Department of Information Systems, University of Münster, Münster, Germany
malte.moeser@uni-muenster.de

Abstract. Bitcoin as deployed today does not scale. Scalability research has focused on two directions: (1) redesigning the Blockchain protocol, and (2) facilitating 'off-chain transactions' and only consulting the Blockchain if an adjudicator is required. In this paper we focus on the latter and provide an overview of Bitcoin payment networks. These consist of two components: payment channels to facilitate off-chain transactions between two parties, and the capability to fairly exchange bitcoins across multiple channels. We compare Duplex Micropayment Channels and Lightning Channels, before discussing Hashed Time-Locked Contracts which enable Bitcoin-based payment networks. Finally, we highlight challenges for route discovery in these networks.

1 Introduction

The Bitcoin community fears that Bitcoin [11], the 'Money for the Internet' and currently the most popular cryptocurrency, cannot scale to meet future demand. Today, the network can process 3.3–7 tps (transactions per second) due to an artificial cap of 1 MB blocks and recent research [5] highlights that 90 % of the network can only achieve an effective throughput of up to 27 tps. This is dwarfed by established payment providers such as Visa, which facilitates about 2,000 tps, with a peak capacity of 56,000 tps [21].

Bitcoin's capacity limitations are increasingly felt by users in the form of delayed transaction processing and rising transaction fees. Users currently pay about 3 to 7 US cents per transaction (independent of the amount transferred). The costs of sending a transaction could continue to rise as competition for space in the Blockchain increases and the protocol's monetary policy continually reduces the minting of new coins that rewards 'miners' for securing the network.

A simple short-term fix to increase Bitcoin's capacity would be to increase the maximum block size, allowing more transactions to be included in each block. While multiple such proposals exist [1,8,12], none have actively been adopted as the community cannot agree if the size of blocks should be increased at all, incrementally, dynamically or whether an artificial cap is required at all.

Long-term research has focused on two directions to improve scalability:

J.K. Liu and R. Steinfeld (Eds.): ACISP 2016, Part I, LNCS 9722, pp. 57–76, 2016.
DOI: 10.1007/978-3-319-40253-6_4

1. Redesigning the underlying Blockchain protocol to support more transactions per second [2,7,9,18], and
2. Facilitating 'off-chain transactions' where transactions are only committed to the Blockchain if an adjudicator is required [6,14].

This paper explores the latter direction and investigates payment networks that facilitate off-chain transactions, using the Blockchain only as a settlement system. Payment networks are attractive as they require only two transactions to be committed to the Blockchain in order to represent all intermediary transactions, they allow bitcoins to be routed across a series of Payment Service Providers (PSPs) without any trust-assumptions, and they provide double-spending prevention as any payment requires the recipients cooperation. Most importantly, depositors using payment networks are guaranteed to receive their fair share of a channel's balance at any point in time.

Payment networks can be split into two major components: payment channels that sends bitcoins between two parties off-chain and the capability to fairly exchange bitcoins across two or more of such channels. In this paper, we first present important concepts of Bitcoin and its underlying lock time rules that enable simple unidirectional and bidirectional payment channels (Sect. 2). Then, we explore and compare more complex payment channel constructions such as Duplex Micropayment Channels and Lightning Channels (Sect. 3). After that, we discuss Hashed Time-Locked Contracts that guarantee the fair exchange of bitcoins across two or more channels (Sect. 4) and highlight challenges for route discovery in these payment networks (Sect. 4.3).

2 Background

We first introduce the necessary background about Bitcoin and its transaction lock time rules used by payment networks. Then, we present how to establish and send bitcoins in basic payment channels with an untrusted counterparty.

2.1 Bitcoin

Bitcoin is a decentralized transaction system, based on a peer-to-peer network and a probabilistic consensus mechanism [11]. In Bitcoin, addresses are the equivalent of accounts in traditional payment systems, used to send and receive funds. An address is a cryptographic hash of a public key, and the corresponding private key is used to spend coins by digitally signing transactions. Addresses provide users with a degree of anonymity and it is recommended practice not to reuse them when receiving bitcoins to preserve the user's privacy. Technically, bitcoins are not strictly sent to a Bitcoin address. Instead, bitcoins are associated with redemption criteria specified in a Forth-like script language. Authorising a transfer of coins requires satisfying these criteria. The two most popular script programs are the 'Pay to public key hash' that requires a signature corresponding

to an address, and the 'Pay to script hash' that, among others, enables multi-signature addresses which require a threshold of m signatures from n public keys.

Coins are transferred via transactions which have one or more inputs and one or more outputs. Each output specifies the number of bitcoins sent and the script program, whereas each input provides a reference to a previous transaction's output and the redeem script (e.g., a corresponding signature) that satisfies the output's spending conditions. Transactions cannot send more bitcoins than provided in the inputs, and there are no rules on how the bitcoins are split amongst the outputs. Also, transaction signers can choose to include a fee that is deducted from the available bitcoins to spend.

Transactions are serialized in blocks that form a public ledger called the Blockchain. Bitcoin's blockchain is updated approximately every ten minutes, each update corresponding to a new block that contains recent transactions. For a block to be valid, it must be cryptographically linked with a previous block and provide a solution to a computationally difficult 'puzzle'. Miners are rewarded for each block with a fixed amount of bitcoins and all transaction fees paid by transactions in the block. The chain of blocks that solves the most computationally difficult puzzles is accepted by the network as the genuine Blockchain and the position of a block in this chain is called 'height'. Due to the lottery process of solving the puzzle, more than one block can be found at the same time, leading to uncertainty at the tip of the chain. A transaction is therefore regarded as 'confirmed' once the block that includes it has achieved a depth of at least six blocks from the tip of the chain [11].

The time dimension given by the block heights and timestamps enables two lock time rules that are fundamental to payment channels:

Absolute Lock Time ensures an entire transaction[1] or a child transaction that is spending an output of a parent transaction[2] cannot be accepted into the Blockchain until a specified absolute block height k (or time) in the future,

Relative Lock Time ensures a child transaction that is spending an output of a parent transaction cannot be accepted into the Blockchain until the parent transaction has achieved a relative depth of λ blocks[3].

2.2 Payment Channel Establishment

A payment channel allows two parties to send numerous payments to each other. Instead of settling all transactions directly on the Blockchain, a payment channel only requires two transactions: one to open the channel, and one to close it and settle the final balance. The cornerstone of payment channels is depositing bitcoins into a multi-signature address controlled by both parties and having the guarantee that all bitcoins are eventually refunded at a mutually agreed time if the channel expires.

[1] The `nLockTime` field of the transaction.
[2] The output's script contains the `OP_CHECKLOCKTIMEVERIFY` opcode.
[3] The output's script contains the `OP_CHECKSEQUENCEVERIFY` opcode.

Spilman [19] proposed the first payment channel establishment protocol without the need to trust the counterparty. This protocol has a *Funding Transaction* that stores the depositor's bitcoins and requires the authorisation of both parties to spend, and a *Refund Transaction* that returns the funds to the depositor if no payments have been authorized or the counterparty abandons the protocol. The lock time on the refund determines the lifetime of the payment channel. To establish the channel, the *Funding Transaction* is created by the depositor and remains unpublished until she received valid signatures for the *Refund Transaction* from the counterparty.

Another possibility to realise refunds is through *Non-Interactive Time-Locked Refunds*, that recently became available in Bitcoin with the activation of BIP 65 [20]. *Non-Interactive Time-Locked Refunds* account for the channel timeout directly in the multi-signature output of the *Funding Transaction*, which means that dedicated refund transactions are no longer necessary. Once the *Absolute Lock Time* specified in the output's script has expired, the refund condition can be redeemed without the cooperation of the counterparty. While the output can only refund a single party, refunding multiple parties is possible with dedicated multi-signature outputs representing each participant's deposit.

In practice, the use of payment channel protocols is currently hindered by the problem of transaction malleability and the infeasibility to build upon unsigned transactions. Transaction malleability allows a co-signer or an external third party to change the identification hash of a transaction before it is accepted into the Blockchain. This is a problem for contracts that sign child transactions before the parent transaction, whose outputs are being spent, is included in the Blockchain. If a modified parent transaction is accepted into the Blockchain, then all pre-signed child transactions become invalid. Furthermore, it is impossible to build upon unsigned transactions as adding signatures to a transaction changes its identification hash. BIP 66 [22] has been deployed to prevent third-party signature malleability, but co-signer and other types of malleability remain. BIP 141 [10], however, solves these malleability issues and allows to build upon unsigned transactions, and is likely to be deployed soon.

2.3 Basic Payment Channels

We introduce unidirectional and bidirectional channels that leverage the *Funding Transaction*'s multi-signature output. The establishment protocol in Sect. 2.2 is used to set up both channels. Bitcoins are sent using subsequent *Payment Transactions* that have two outputs to send each party their respective bitcoins.

Unidirectional channels were first implemented by Corallo in Bitcoinj [3] to allow a customer to send incremental payments to a merchant. Each payment has two outputs: the first increases the amount of bitcoins sent to the merchant, and the second returns change to the customer. This introduces the *Replace by Incentive* rule as the merchant only signs and broadcasts the latest *Payment Transaction* that sends them the most bitcoins. Payments can be made until the channel expires or the whole deposit has been transferred to the merchant.

Bidirectional channels require the *Payment Transaction* to be associated with an *Absolute Lock Time*. Each incremental payment decrements the lock time by a safety margin Δ that represents the expected time for transactions to be accepted into the Blockchain. This introduces the *Replace by Timelock* rule as the latest *Payment Transaction* is guaranteed to be accepted into the Blockchain before any previously authorised transaction. Each payment requires both parties to exchange signatures and reduces the channel's lifetime.

3 Proposed Payment Channel Protocols

In this section, we outline two proposals for payment channels. The first one is called Duplex Micropayment Channels, due to Decker and Wattenhofer [6]. It extends the number of transactions that can occur within the lifetime of a bidirectional channel. The second one is called Lightning Channels, proposed by Poon and Dryja [14], and allows the channel to remain open indefinitely.

3.1 Duplex Micropayment Channels

Decker and Wattenhofer propose Duplex Micropayment Channels which enable bidirectional payment channels with a finite lifetime [6]. Their scheme builds upon an initial *Funding Transaction* with deposits from two parties A and B. It consists of two *Unidirectional channels* C_{AB}, C_{BA} that together allow bidirectional payments, and an *Invalidation Tree* that sets the minimum lifetime of the channel and is responsible for resetting the bitcoins available to spend in both channels C_{AB}, C_{BA}. This reset is necessary as Alice may receive 1 BTC from Bob in the unidirectional channel C_{BA}, but her unidirectional channel to Bob C_{AB} has exhausted its supply of bitcoins. To continue bidirectional payments it is necessary to refresh C_{AB} with Alice's 1 BTC.

The core idea of Duplex Micropayment Channels is to apply the *Replace by Timelock* rule (cf. Sect. 2.3) using a tree of timelocked transactions instead of a single transaction. This structure is called an *Invalidation Tree*, exemplarily shown in Fig. 1. The nodes in the tree represent Bitcoin transactions; each edge corresponds to the spending of the previous node's output and is signed by both parties. Each transaction $T_{d,k}$ has a depth d in the tree and is associated with an *Absolute Lock Time* k. Nodes in the active branch $(T_{1,k}, T_{2,k}, ..., T_{d,k})$ have *Absolute Lock Times* that are less than previously authorised branches. This guarantees that the active branch is accepted into the Blockchain before previously authorised branches.

The leaf node $T_{d,k}$ on the active branch has two outputs to represent each unidirectional channel C_{AB} and C_{BA}. Each output can be spent if either of the following conditions is satisfied:

1. The first condition requires the signature of both parties to authorise a *Payment Transaction*.

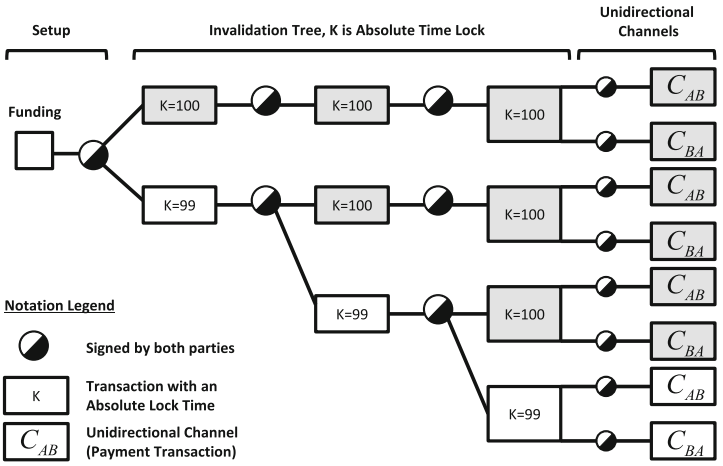


Fig. 1. Duplex Micropayment Channels. A *Funding Transaction* is stored in the Blockchain. All payments occur in the unidirectional channels. If either channel becomes exhausted, then a new branch is created in the *Invalidation Tree* with a smaller *Absolute Time Lock* k. This invalidates all previous branches, and resets the balance of both unidirectional channels. For illustration $\Delta = 1$.

2. The second condition requires the signature of the depositor and that the current Blockchain height is greater than an *Absolute Lock Time* $k_{\max}$. This lock time $k_{\max}$ is the maximum waiting time before the bitcoins can be returned to the depositor using a *Refund Transaction*.

Payments are sent in the unidirectional channels C_{AB}, C_{BA} as the sender signs subsequent *Payment Transactions* that have two outputs: the first sends bitcoins to the receiver, while the second returns change to the sender. The *Replace by Incentive* rule implies that the receiver will only sign the *Payment Transaction* that sends them the most bitcoins. If the receiver is unresponsive, the depositor is guaranteed to have their bitcoins refunded once the maximum lifetime of the channel $k_{\max}$ expires. It is possible to reset the balance of both unidirectional channels once a channel has exhausted its supply of bitcoins. Resetting the channels requires replacing the current active branch in the *Invalidation Tree* with a new branch whose leaf node $T_{d,k}$ acts as an anchor for a new set of unidirectional channels.

In the following, we explain how to establish the channel, send bidirectional payments, reset the balance of both *Unidirectional channels* and finally settle the payment channel on the Blockchain.

Channel Establishment. First, both parties combine their funds in an unsigned *Funding Transaction*. Then, they build the first branch $(T_{1,k}, T_{2,k}, \ldots, T_{d,k})$ of the *Invalidation Tree* and exchange signatures for the branch and the *Payment Transactions* that represent the unidirectional channels C_{AB}, C_{BA}. Finally, both parties sign and broadcast the *Funding Transaction*.

The *Absolute Lock Time* $k_{\max}$ of the *Refund Transaction* should exceed the *Absolute Lock Time* of the leaf node $T_{d,k}$ by at least a safety margin Δ. This ensures that the unidirectional channels have enough time to be accepted into the Blockchain before the *Refund Transactions* become available to spend.

Send a Payment. Each payment requires the sender to sign a new *Payment Transaction*. The receiver only signs the transaction that sends them the most bitcoins if they want to settle a dispute on the Blockchain. If either unidirectional channel has exhausted its supply of bitcoins, then both parties must cooperate to reset the balance of both channels before further payments can be made.

Reset Balance. Resetting the balance of the unidirectional channels requires both parties to agree a new active branch in the *Invalidation Tree*. Figure 1 demonstrates several branch replacements. An example includes the current active branch $(T_{1,99}, T_{2,99}, T_{3,99})$ replacing the previous branch $(T_{1,99}, T_{2,99}, T_{3,100})$.

First, both parties find the first node $T_{\alpha,k}$ closest to the leaves that has a greater *Absolute Lock Time* than it's parent node by a safety margin of Δ, otherwise the root node $T_{1,k}$ is chosen.

Second, a new branch is created, starting with the chosen node whose *Absolute Lock Time* is decremented by the safety margin Δ (i.e. $T_{\alpha,k-\Delta}$). Each child node in this new branch is given an *Absolute Lock Time* greater than $k-\Delta$ (e. g., the maximum value initially chosen when the tree was established). As lock times are *transitive*, the reduced lock time of the parent transaction automatically invalidates all previously authorised branches. Note, that when the lock time of the root node $T_{1,k-\Delta}$ is decremented, then the channel's minimum lifetime $k_{\min}$ is also reduced.

Third, both parties exchange signatures for the new active branch. The signatures for the node $T_{\alpha,k-\Delta}$ are only exchanged once both parties have successfully exchanged signatures for the *Payment Transactions* that represent the unidirectional channels C_{AB}, C_{BA} and all of its child nodes $(T_{\alpha+1,k}, ..., T_{d,k})$.

Settle Channel. Both parties cooperate to sign and broadcast a transaction that has no *Absolute Lock Time* and has two outputs to send each party their respective final balance. If both parties cannot cooperate, then either party can broadcast the current active branch for inclusion into the Blockchain once the *Absolute Lock Times* expire. If the receiver in a unidirectional channel does not broadcast the *Payment Transaction* that sends them the most bitcoins, then the depositor waits until $k_{\max}$ to broadcast the *Refund Transaction*.

3.2 Lightning Channels

Poon and Dryja propose bidirectional payment channels called Lightning Channels that can remain open indefinitely [14]. Sending a payment requires the cooperation of both parties to authorise a new channel state that represents each party's new balance, before revoking the channel's current state. The revocation mechanism requires both parties to check the Blockchain periodically to detect

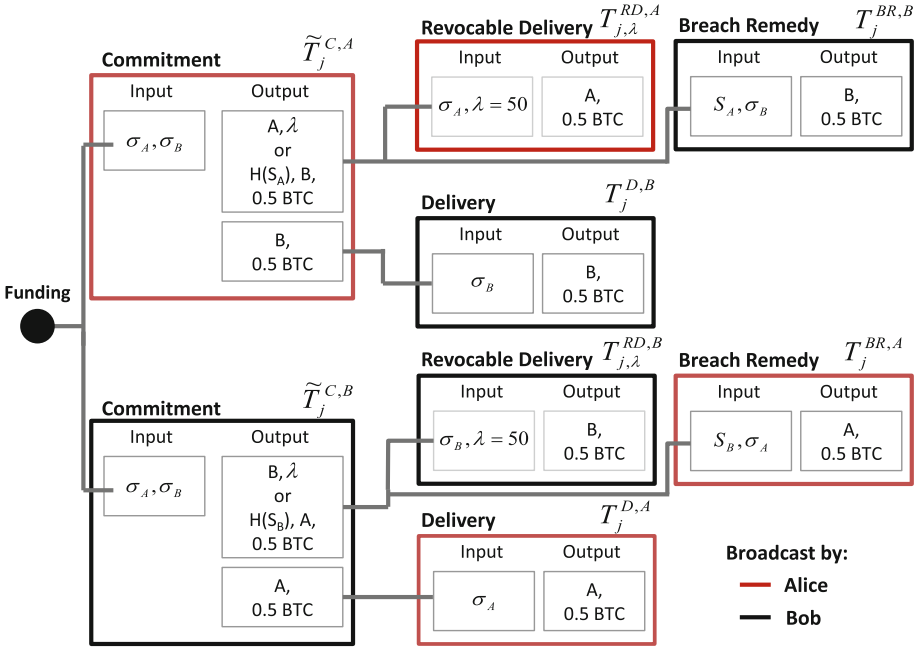


Fig. 2. Lightning Channels. Each party has a *Commitment Transaction* that only they can broadcast. To settle a dispute on the Blockchain, either party can broadcast their *Commitment Transaction* and *Revocable Delivery Transaction* to claim their share of bitcoins. In response, the counterparty broadcasts their *Delivery Transaction* to receive their share of bitcoins. If a *Commitment Transactions* has previously been revoked, the counterparty can broadcast the *Breach Remedy Transaction* to steal all bitcoins in the channel. (Color figure online)

if a previously revoked channel state has been submitted. If a revoked state is detected, the counterparty that did not broadcast the revoked state can issue a penalty to claim all bitcoins in the channel. Initially, this revocation was performed by exchanging signatures for the penalty transaction. An improvement proposed by Back is outlined in [16] to use the pre-image S of a revocation hash $H(S)$. This revocation hash is used in our description of Lightning Channels.

To describe the protocol, we use the following notation: transactions are denoted as $T_{j,\lambda}^{\beta,X}$, where β is an acronym for the transaction's name, X is the party that can broadcast the transaction (i.e. it requires party X's signature to complete the transaction), j is the number of updates that have occurred in the channel and λ is an optional *Relative Lock Time*. Furthermore, T only requires the broadcaster to sign, $\tilde{T}$ requires both parties to sign and $\sigma_{X,j}^{\beta}$ represents party X's signature for the j^{th} transaction β.

Figure 2 presents the underlying transaction structure of Lightning Channels. The channel relies upon an initial *Funding Transaction* $\tilde{T}^F$ that is signed by both parties and deposits bitcoins into a multi-signature address. The symmetric structure of the scheme provides each party with the following transactions:

Channel Establishment:
$A \rightarrow B$: Revocation hash $h_{A,j}$ and her transaction input π
$B \rightarrow A$: Unsigned $\tilde{T}^F$, signature $\sigma^C_{B,j}$ for $T^{C,A}_j$, revocation hash $h_{B,j}$
$A \rightarrow B$: Signature $\sigma^C_{A,j}$ for $T^{C,B}_j$, signature σ^F_A for T^F
$B \rightarrow N$: Broadcast $\tilde{T}^F$ to the Bitcoin network.
Send a Payment:
$A \rightarrow B$: New revocation hash $h_{A,j+1}$
$B \rightarrow A$: Signature $\sigma^C_{B,j+1}$ for new $T^{C,A}_{j+1}$, new revocation hash $h_{B,j+1}$
$A \rightarrow B$: Signature $\sigma^C_{A,j+1}$ for new $T^{C,B}_{j+1}$, pre-image $S_{A,j}$ of previous revocation hash $h_{A,j}$
$B \quad A$: Pre-image $S_{B,j}$ of previous revocation hash $h_{B,j}$

Fig. 3. Sequence of messages exchanged between two parties A and B to establish a Lightning Channel and send payments.

The *Commitment Transaction* $\tilde{T}^{C,X}_j$ spends the multi-signature output and therefore requires signatures from both parties. Each party receives the counterparty's signature in advance. The first output sends bitcoins to the broadcaster, while the second outputs sends bitcoins to the counterparty. Spending the second output only requires the counterparty's signature, but spending the first output is only possible if either of the following two conditions is fulfilled:

1. The first condition requires a signature from the broadcaster and has a *Relative Lock Time* such that the *Commitment Transaction* $\tilde{T}^{C,X}_j$ needs to achieve a depth of λ before the output is spendable.
2. The second condition requires the pre-image $S_{X,j}$ of a revocation hash $h_{X,j} = H(S_{X,j})$ and a signature from the counterparty.

These conditions allow *Commitment Transactions* to be revoked by revealing the pre-image $S_{X,j}$ to the counterparty. The normal payout to the broadcaster is delayed by a *Relative Lock Time* λ to allow the counterparty to claim the funds if the transaction has previously been revoked. Revoking a *Commitment Transaction* allows both parties to update the channels while ensuring that no revoked transaction will ever be published.

The *Revocable Delivery Transaction* $T^{RD,X}_{j,\lambda}$ requires the signature of the broadcaster, and sends bitcoins to the broadcaster. It can only be broadcast once the broadcaster's *Commitment Transaction* has achieved a depth of λ in Bitcoin's blockchain.

The *Delivery Transaction* $T^{D,X}_{j,\lambda}$ requires the signature of the counterparty that did not broadcast the *Commitment Transaction* and immediately sends the counterparty their share of bitcoins.

The *Breach Remedy Transaction* $T^{BR,X}_j$ requires the signature of the counterparty that has not broadcast the revoked *Commitment Transaction* and the pre-image $S_{X,j}$ of the *Commitment Transaction*'s revocation hash. It can only be broadcast once a revoked *Commitment Transaction* has been accepted into the Blockchain. No *Relative Lock Time* is associated with this transaction to allow the counterparty to issue the penalty immediately.

In the following and in Fig. 3, we explain how to establish the channel, send bidirectional payments, and finally settle the payment channel on the Blockchain.

Channel Establishment. Alice sends Bob her revocation hash $h_{A,j}$ and a transaction input π_A for the *Funding Transaction* $\tilde{T}^F$. Bob responds with an unsigned *Funding Transaction* $\tilde{T}^F$ that includes inputs of both parties, a signature $\sigma^C_{B,j}$ for Alice's *Commitment Transaction* $T^{C,A}_j$ and his revocation hash $h_{B,j}$. Then, Alice sends a signature $\sigma^C_{A,j}$ for Bob's *Commitment Transaction* $T^{C,B}_j$ and a signature σ^F_A for the *Funding Transaction* T^F. Finally, Bob signs and broadcasts the *Funding Transaction* $\tilde{T}^F$ to the network.

Send a Payment. Sending a payment requires the cooperation of both parties to authorise a new set of transactions to represent the channel's new balance before invalidating the current set of transactions.

Alice sends Bob a new revocation hash $h_{A,j+1}$. Bob responds with a signature $\sigma^C_{B,j+1}$ for Alice's new *Commitment Transaction* $T^{C,A}_{j+1}$, and his new revocation hash $h_{B,j+1}$. Alice checks that the bitcoins are correctly distributed to each party in $T^{C,A}_{j+1}$ before sending her signature $\sigma^C_{A,j+1}$ for Bob's new *Commitment Transaction* $T^{C,B}_{j+1}$. Bob then also checks that the bitcoins are correctly distributed to each party. Once the channel's new state has been authorised, Alice sends the pre-image $S_{A,j}$ of her revocation hash to Bob, and this pre-image revokes her current *Commitment Transaction* $T^{C,A}_j$. Bob checks if the pre-image correctly revokes Alice's current *Commitment Transaction* before sending Alice the pre-image $S_{B,j}$ for his revocation hash $h_{B,j}$. Finally, Alice checks if this pre-image revokes Bob's current *Commitment Transaction* $T^{C,B}_j$.

Settle Channel. To settle the channel, both parties cooperate to sign and broadcast a transaction that sends each party their share of bitcoins. If either party does not cooperate, then the dispute is settled on the Blockchain. In a dispute, either party broadcasts their most recent *Commitment Transaction* $\tilde{T}^{C,A}_j$ and *Revocable Delivery Transaction* $T^{RD,A}_{j,\lambda}$. The counterparty must then broadcast their *Delivery Transaction* $T^{D,B}_j$ to receive their share of the coins.

3.3 Comparison of Duplex Micropayment and Lightning Channels

This section provides a comparison of Duplex Micropayment Channels and Lightning Channels. We focus on the computation, storage and network access required for both schemes before highlighting if resource-limited participants can outsource responsibility to a trusted third party. Finally, we discuss the total number of transactions that can be facilitated.

Blockchain Privacy. A new pair of addresses A_1, B_1 is generated by both parties for the *Funding Transaction*'s multi-signature output. If both channels are closed cooperatively, then A_1, B_1 can be reused in the closing transaction that sends each party their final balance. In terms of privacy, an external Blockchain

Table 1. The number of signatures required for each step in Duplex Micropayment Channels and Lightning Channels. Also, d represents each node in the *Invalidation Tree* and α is the number of replaced nodes in the *Invalidation Tree*.

	Set up	Payment	Reset	Settle (Co-op)	Settle (Dispute)
Duplex	$(d+2) \times 2$	1	$(\alpha+1) \times 2$	1×2	1×2
Lightning	2×2	1×2	0	1×2	3

observer can see the number of bitcoins each address received, but not identify which receiving address A_1, B_1 corresponds to which depositing address A_0, B_0.

Throughout the lifetime of both schemes, A_1, B_1 can be reused in each intermediary transaction to minimise computing addresses. If a dispute is settled on the Blockchain, then a Duplex Micropayment Channel achieves the same privacy as using a single transaction to close the channel. While Lightning Channels also provide these privacy guarantees, they allow to identify which address A_1, B_1 raised the dispute by identifying the address that received bitcoins in the *Revocable Delivery Transaction*.

In Duplex Micropayment Channels, both parties can have a different pair of addresses for each unidirectional channel. By inspecting the outputs of both *Payment Transactions* it is not possible to derive which addresses belong to the same owner. It is also not possible to determine which party raised the dispute using the transactions from the Blockchain only. The parties in Lightning Channels can compute 3 addresses to be used for both sets of *Commitment Transactions*. For example, Alice's *Commitment Transaction*'s first output sends bitcoins to either A_1 or B_1, and the second output sends bitcoins to B_2. Her *Revocable Delivery Transaction* sends bitcoins to A_2 and Bob's *Delivery Transaction* sends bitcoins to B_3. Here, it is still possible to determine which addresses raised the dispute and also the final share of bitcoins each set of addresses received.

The number of signatures required in each payment channel is shown in Table 1. To establish the channel, both schemes require each party to sign a *Funding Transaction*. In Duplex Micropayment Channels each party must also sign d nodes in the *Invalidation Tree* and a *Payment Transaction* representing the unidirectional channel, whereas Lightning Channels require each party to sign an additional *Commitment Transaction*.

To send a new payment, the Duplex Micropayment Channels require a single signature from the sender. If either unidirectional channel has exhausted its supply of bitcoins, then both parties cooperate to reset the balance of both channels. This reset requires each party to sign α replacement transactions in the *Invalidation Tree* and an additional signature for the new *Payment Transactions* that represent the unidirectional channels. Lightning Channels always require a single signature from both parties to authorise a new pair of *Commitment Transactions*. This has implications for popular hubs as Duplex Micropayment Channels do not require the hub's involvement to receive bitcoins (except to perform a reset), while the hub must sign every payment in Lightning Channels.

In both schemes parties can cooperatively close the channel by signing a transaction that settles the final balance. However, they must sign the remaining intermediary transactions if the channel is not closed cooperatively. The unsigned transactions in Duplex Micropayment Channels include the *Payment Transaction*, or a *Refund Transaction* if the counterparty does not sign their *Payment Transaction*. In Lightning Channels, either party can broadcast their *Commitment Transaction* and sign a *Revocable Delivery Transaction* to claim their bitcoins. In response, the counterparty must sign the *Delivery Transaction* to receive their share of bitcoins. If the *Commitment Transaction* was previously revoked, then the counterparty must instead sign a *Breach Remedy Transaction*.

Local Storage Requirements. In Duplex Micropayment Channels, both parties store $d + 1$ transactions which include the current active branch in the *Invalidation Tree* and the most recently received *Payment Transaction* from the counterparty. In Lightning Channels, each party stores the pre-image for every revocation hash used by the counterparty and the current *Commitment Transaction*. To prevent the need to brute-force the revocation hash using all stored pre-images, parties should also store a mapping between each pre-image and corresponding revocation hash.

Storage in the Blockchain. When both parties cooperate, both schemes only require 2 transactions, the *Funding Transaction* and a *Settlement Transaction*, to be committed to the Blockchain. In the worst case, when parties do not cooperate, Duplex Micropayment Channels require $1 + d + 2$ transactions. The d transactions represent the *Invalidation Tree*'s active branch, plus the *Funding Transaction* and 2 additional transactions representing either *Payment Transactions* for the unidirectional channels or their *Refund Transactions*. Lightning Channels always require 4 transactions. Besides the *Funding Transaction*, this includes a *Commitment Transaction*, the corresponding *Delivery Transaction* and either a *Breach Remedy Transaction* or a *Revocable Delivery Transaction*.

Frequency of Access to the Blockchain. Duplex Micropayment Channels rely on the *Replace by Timelock* rule to ensure that only the latest authorised branch in the *Invalidation Tree* is accepted into the Blockchain. Therefore, after establishing the channel, neither party needs to access the Blockchain until the channel has expired. Then, however, it is important to push the d transaction in the latest branch and the *Payment Transaction* that sends the party their bitcoins as soon as they become valid. Lightning Channels require both parties to periodically scan the Blockchain for previously revoked transactions. The frequency of monitoring is based on the mutually agreed *Relative Lock Time* that delays the *Revocable Delivery Transaction*'s acceptance into the Blockchain.

Outsourcing Responsibilities is possible in both schemes to reduce the burden for resource-limited parties. In Duplex Micropayment Channels, a trusted third party can be responsible for broadcasting the active branch of the *Invalidation Tree* and the *Payment Transaction* that sends the party their bitcoins once the channel has expired. The branch must be broadcast within the safety-margin

Δ time span, otherwise previously authorised branches become spendable. In Lightning Channels, the trusted third party is provided with signed *Breach Remedy Transactions*, pre-images of revocation hashes and the identification hash of previously revoked *Commitment Transactions*. It is then responsible for monitoring the Blockchain for the previously revoked *Commitment Transactions* and broadcasting the corresponding *Breach Remedy Transactions*.

The total number of transactions that can occur in Duplex Micropayment Channels is based on the number of bitcoins sent, the frequency of resets to replenish the unidirectional channels, the depth of the tree, and the *Absolute Lock Time* associated with each node. In comparison, the only limitation for Lightning Channels is the bidirectional activity between both parties.

4 Routing Payments

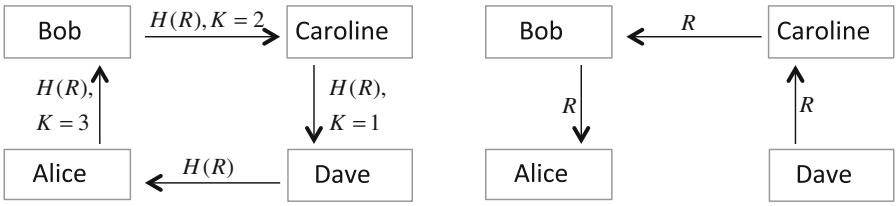
We present how to fairly exchange bitcoins across several payment channels with Hashed Time-Locked Contracts (HTLCs) before discussing how to mitigate routing interruptions and the challenge of route discovery on the payment network.

4.1 Hashed Time-Locked Contract (HTLC)

Hashed Time-Locked Contracts fairly exchange bitcoins across several payment channels. They allow two parties to open channels with separate Payment Service Providers (PSP), and then construct a route of payment channels that connects both channels. If we have three channels, Alice $\rightarrow$ Bob, Bob $\rightarrow$ Caroline and Caroline $\rightarrow$ Dave, then Alice can send bitcoins to Dave, via Bob and Caroline. This fair exchange guarantees that intermediary hops receive their bitcoins, once they have sent their bitcoins to the next hop.

Figure 4 outlines the exchange of messages required to fairly exchange bitcoins across all intermediary channels. Once a route has been selected, the final receiver (Dave) provides the initial sender (Alice) with an HTLC hash $H(R)$. The initial sender commits bitcoins in a payment channel shared with their PSP under the condition that the pre-image R is revealed within k blocks. Each intermediary along the route decrements the lock time k and repeats this commitment with the next hop. The lock time k is decremented by ω to provide the intermediary a timespan for their bitcoins to be sent to the next hop and to allow them to claim their bitcoins from the previous hop. The last hop is the final receiver, who receives the payment amount contingent upon providing R.

The final receiver can claim the bitcoins by creating a transaction that reveals R and spends the HTLC output. Revealing R then allows the next party to also claim the bitcoins from their previous hop. However, it is not necessary to claim the outputs using Bitcoin's blockchain. Instead, both parties may simply agree to update their shared channel to reflect the new balance. This can be done along the route: every pair of participants updates their channels until the initial sender's bitcoins are taken.



(a) Setting up HTLC (b) Synchronizing HTLC balances

Fig. 4. (a) The final receiver provides the initial sender with the HTLC hash $H(R)$. This is shared along the HTLC route and the HTLC transfer's *Absolute Lock Time* k is decremented for each hop. (b) Outlines how the pre-image R is revealed in the reverse order for each hop to claim their bitcoins once the final receiver is given $H(R)$. For illustration the decrement time is $\omega = 1$.

In Duplex Micropayment Channels a new HTLC output is either added as part of a new *Payment Transaction* in the unidirectional channel or as an additional output to the leaf node $T_{d,k}$ in the *Invalidation Tree* (cf. Fig. 5). This HTLC output is claimable if either of the following two conditions is satisfied:

1. The first condition requires a signature from both parties.
2. The second condition requires a signature from both parties[4] and the pre-image R of the HTLC hash $H(R)$.

This HTLC output can be spent using either of the following three transactions. The *HTLC Refund Transaction* guarantees that the bitcoins are returned to the sender by k_{refund} and satisfies the first condition of the HTLC output. The *HTLC Settlement Transaction* requires the pre-image R of the HTLC hash and sends bitcoins to the counterparty. This transaction has an *Absolute Lock Time* k_{settle} to delay the receiver claiming the bitcoins to ensure that the HTLC output can later be cancelled if both parties agree. This transaction spends the second condition of the HTLC output. The *HTLC Forfeiture Transaction* has no lock time and is signed by both parties to cancel the HTLC transfer. It spends the first condition of the HTLC output and guarantees that the bitcoins are returned to the sender, even if the pre-image R of the HTLC hash is revealed.

It is the responsibility of the initial sender to compute the lock time k_{refund} for each hop along the route. The sender needs to ensure that each hop's k_{refund} is greater than the hop's chosen k_{settle} (i.e. $k_{\text{refund}} > k_{\text{settle}}$). Both lock times must also be greater than the hop's leaf node $T_{d,k}$ lock time k_{leaf} (i.e. $k_{\text{settle}} > k_{\text{leaf}}$). This is required as a refund or settlement cannot happen until the leaf node $T_{d,k}$ is included in Bitcoin's blockchain. Also, a timespan between the leaf node's lock time k_{leaf}, and the settlement lock time k_{settle} is required to allow the hop to cancel the HTLC transfer using a *HTLC Forfeiture Transaction*. This means that the bitcoins are potentially locked for the entire lifetime of the channel.

[4] The sender must have a different Bitcoin address for each condition in the HTLC output, otherwise the receiver can use the signature to satisfy either condition.

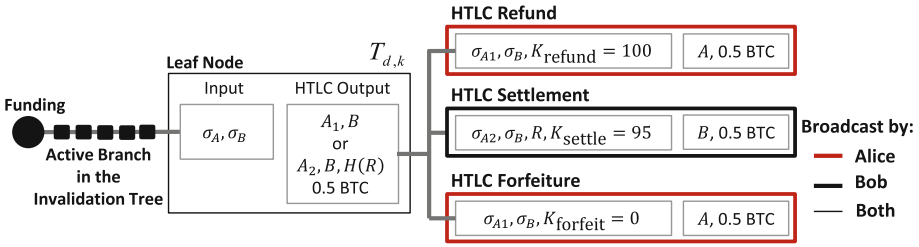


Fig. 5. The HTLC is attached as an additional output to the *Invalidation Tree*'s leaf node. We have omitted unidirectional channels for space. The *HTLC Refund Transaction* guarantees that the bitcoins are returned to the sender if R is not revealed by block k_{refund}, the *HTLC Settlement Transaction* sends bitcoins to the receiver if the pre-image R of the HTLC hash is revealed by block k_{settle}, and the *HTLC Forfeiture Transaction* can later be signed to cancel the transfer. (Color figure online)

The initial sender's k_{refund} is passed to the next hop on the route, who should decrement k_{refund} by ω for use in their channel. This repeats as each hop passes their k_{refund} to the next hop. The initial sender's k_{refund} must therefore take into account the largest leaf node's lock time k_{leaf} in the route. In the worst case, if the largest k_{leaf} is associated with the final receiver's channel, then the lock time k_{refund} for all hops along the route must also be larger than the largest k_{leaf}.

Finally, the hash $H(R)$ and the pre-image R can be discarded once the HTLC transfer has been forfeited or a new active branch in the *Invalidation Tree* is mutually agreed to cancel the HTLC transfer.

In Lightning Channels, a new HTLC output is associated with both *Commitment Transactions* in a new channel state (cf. Fig. 6). The broadcaster of a *Commitment Transaction* is restrained by a *Relative Lock Time* λ before she can claim the bitcoins in the HTLC output. This provides the counterparty an opportunity to claim the bitcoins. We explain the role of λ once the commonality of the HTLC output's redemption criteria for each *Commitment Transaction* has been presented. The bitcoins can be claimed if either of the following three conditions is satisfied:

1. The first condition requires a signature from the receiver and the pre-image R of the HTLC hash $H(R)$.
2. The second condition requires a signature from the sender once the *Absolute Lock Time* k_{refund} has expired.
3. The third condition requires a signature from the counterparty that did not broadcast the revoked *Commitment Transaction*, and the pre-image S of the broadcaster's revocation hash $H(S)$.

The broadcaster of a *Commitment Transaction* must consider both the *Absolute Lock Time* k_{refund} that dictates when the HTLC transfer expires, and the *Relative Lock Time* λ which requires the broadcaster's *Commitment Transaction* to achieve a depth of λ blocks in the Blockchain before the broadcaster can claim their bitcoins. To illustrate the additional burden of λ for either party's

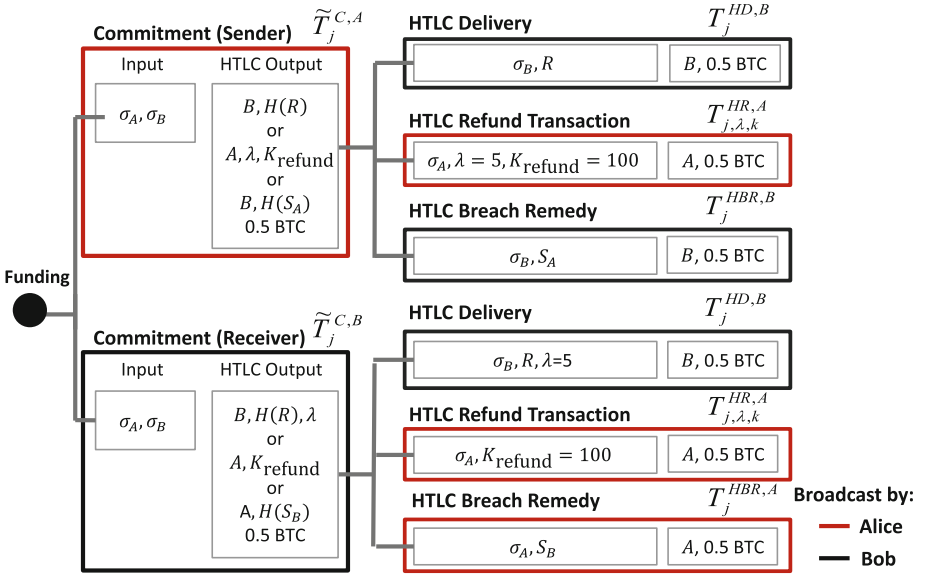


Fig. 6. The HTLC is an additional output in both *Commitment Transactions* and we have omitted the other outputs for space. The delivery sends bitcoins to the receiver if they know R, the refund guarantees that the bitcoins are returned to the sender, and the breach remedy sends bitcoins to the counterparty whom did not broadcast a revoked *Commitment Transaction*. (Color figure online)

redemption criteria, we consider the sender's and receiver's *Commitment Transaction* in turn.

The sender only broadcasts their *Commitment Transaction* to claim a refund using the *HTLC Refund Transaction* if the HTLC transfer expires, satisfying the second condition. To take the delay caused by λ into account, the sender's *Commitment Transaction* must be accepted into the Blockchain at block height $k_{\text{refund}} - \lambda$ for the *HTLC Refund Transaction* to become spendable at the correct time. If the refund is not claimed at the correct time, then R might be later revealed, and the sender is not guaranteed to receive the bitcoins from the previous hop. This λ provides a timespan for the counterparty to claim the bitcoins using the pre-image R of the HTLC hash $H(R)$, satisfying the first condition, or with the pre-image S_A of the revocation hash $H(S_A)$, satisfying the third condition.

The receiver only broadcasts their *Commitment Transaction* if they know the pre-image R of the HTLC hash. The bitcoins are redeemed using the *HTLC Delivery Transaction* that satisfies the first condition. To take the delay caused by λ into account, the receiver's *Commitment Transaction* must be accepted into Bitcoin's blockchain at block height $k_{\text{refund}} - \lambda - \Delta$. This makes certain that the *HTLC Delivery Transaction* has a safety margin Δ timespan to be accepted into Bitcoin's blockchain before the HTLC transfer expires. This λ provides a

timespan for the counterparty to claim the bitcoins if the transfer has expired or if the pre-image S_B of the receiver's revocation hash $H(S_B)$ has been revealed.

The initial sender is responsible for computing the lock time k_{refund} in advance for each hop. This k_{refund} is passed along the route, and each hop decrements k_{refund} by ω for use in their channel. However, the initial sender's k_{refund} must take into account the hop with the largest *Relative Lock Time* λ. In the worst case, if the largest λ is associated with the final receiver's channel, then the lock time k_{refund} for all channels must also include the largest λ. This potentially locks the bitcoins for a long period of time as resource-limited parties may require a large λ as it dictates the frequency in which each party must monitor the Blockchain. Also, the pre-image R can be discarded once both parties have exchanged the pre-images S_A, S_B of the revocation hashes $H(S_A), H(S_B)$.

4.2 Routing Interruptions

The routing is interruptible if an intermediary is unresponsive after accepting the HTLC transfer. The initial sender and final receiver should wait until the HTLC transfer expires before reattempting the transfer. To overcome this issue, Poon has proposed a rollback protocol [13] that allows the final receiver to refund the initial sender using a second route. This allows the initial sender to reattempt the HTLC transfer, and if the intermediary returns, the initial sender's bitcoins are refunded using the second route. This rollback is only performed if the final receiver has not revealed the first route's pre-image R_1 of the HTLC hash $H(R_1)$.

To perform the rollback, the initial sender provides the final receiver with a new HTLC hash $H(R_2)$ and an *Absolute Lock Time* k_{expire} that represents the expiry time for the initial sender's HTLC transfer with their immediate hop in the first route. The final hop in the second route sends the initial sender the refunded bitcoins and must have an expiry time k_{refund} that is greater than k_{expire}. This provides a timespan for the initial sender's bitcoins to be claimed in the first route, and for them to receive the refund in the second route.

Finally, the receiver commits bitcoins to the next hop under the condition that the pre-images of the first route's HTLC hash $H(R_1)$ and the sender's HTLC hash $H(R_2)$ are revealed. Each hop decrements the lock time $k_{\max}$ by ω, and commits bitcoins to the next hop if R_1, R_2 are revealed. The final hop is the sender who has knowledge of R_2, but can only claim the bitcoins if R_1 is revealed from the first HTLC transfer.

4.3 Challenges for Route Discovery

We now discuss the challenges that payment networks face for route discovery. These challenges include the type of connections available for routing bitcoins, the topology of the network, the difficulty in building reputation systems to help assess the risk of other nodes and the financial incentives for routing bitcoins.

Node connections on the network rely on the user's role. End users might exclusively connect to PSPs who are responsible for establishing channels with other PSPs to find routes on the network. How these channels are funded will also

differ as end users might fund the channel with the PSP, while the PSP to PSP channels are mutually funded. Also, the total number of bitcoins in a channel restricts the bitcoins an end user can receive. For example, if the user and PSP share a channel $U \leftrightarrow P$ in which the PSP has a balance of x bitcoins, then the user can only receive up to x bitcoins without establishing a new channel. The receiver must send bitcoins to the PSP to receive further payments.

Route planning methods include source-routing in which the sender predetermines the payment route, and per-hop routing where only the final destination is given to the network and each hop finds the best route.

The former approach simplifies calculating the fee and lock time for the transfer. It is compatible with onion-routing which only reveals the previous and next hop, and allows the sender to enforce the order in which each node is visited. However, this does not prevent a node using intermediaries to route the bitcoins to the next hop. It might also be possible to identify that the next hop is the final recipient using the remaining fee and lock time.

Per-hop routing outsources the route finding to the network and can adapt to changing network conditions. The final destination is revealed to each node to help them find a route, offering less privacy than source-routing. Nodes could potentially offer low (or negative) fees to encourage others to select them for routing and then sell the transaction data to interested third parties. In per-hop routing, estimating the total fee and maximum lock time is difficult as the route is not known in advance, and mechanisms must be put into place that prevent nodes from taking excessive fees.

The topology of the network hinges on the ease of becoming a PSP and the potential regulation as money transmitters.

A straightforward approach is a hub-and-spoke model [17] with thousands of well-connected hubs who share route-finding information amongst themselves. In this model, end users open channels with hubs, and it is the hub's responsibility to find a route to the receiver's hub. These hubs are expected to be reliable and fast which potentially results in negligible HTLC transfer halts. This is important for Duplex Micropayment Channels as bitcoins can be locked for the channel's remaining lifetime if a transfer halt occurs. Also, the usage of Duplex Micropayment Channels reduces the computational overhead for hubs as they are only required to sign HTLC transfers while sending bitcoins, and not to receive them. In this model, end-users may not be responsible for computing or revealing the pre-image R of the HTLC hash as it would allow them to halt transfers.

The fundamental issue with the hub-and-spoke model is that it may lack Bitcoin's open-membership property. Instead, the community's vision for a payment network is to allow anyone to become a PSP which requires a mesh network and gossip protocol to advertise PSP services. Poon has proposed that Bitcoin's blockchain can aid route discovery as the user can identify which PSPs share channels [17]. If a route has been found, there is no guarantee that the PSPs are reliable due to the nature of open-membership. Lightning Channels are more suitable for unreliable transfers as a failure only temporarily locks the bitcoins.

However, the computation overhead increases as PSPs must sign both sending and receiving HTLC transfers.

To prevent routing failures naturally leads to a reputation system to assess the risk of using a PSP. However, in the event of a failure, it is arguably a non-trivial problem to determine the reason behind the halt due to the private nature of routing and the inability to inspect other payment channels. For example, if the hop $C \rightarrow D$ settles on the Blockchain, it is not possible to determine if Dave had knowledge of the pre-image R, refused to disclose it, or simply raised a dispute to tarnish the reputation of Caroline [15]. Also, it might not be possible to identify which hop halted the transfer when disputes are not settled on the Blockchain. Furthermore, revealing the internal channel state of intermediaries cannot always reveal the reason behind a halt. For example, it is not possible to identify the duration of time before R was disclosed.

Routing fees need to cover the costs for a PSP to operate a secure node. These costs potentially include a 'risk' charge based on the number of bitcoins maintained in their hot wallet[5] to facilitate transfers. The PSP might charge if they are required to deposit bitcoins into the channel for end-users (i.e. merchants) who expect to mostly receive bitcoins. Also, negotiating a fee for routing is not straightforward as some PSPs may fluctuate fees to move funds in a maxed-out channel in a certain direction [4]. How fees are negotiated depends on the topology of the network as PSPs might have direct connections with each hop in the route or rely on fees advertised via a gossip protocol. Finally, the fee structure for end-users can be regular installments or on a per-transfer basis.

5 Conclusion

In this paper we presented an overview of Blockchain-based payment networks, a potential scaling solution for Bitcoin. We compared two prominent proposals, Duplex Micropayment Channels and Lightning Channels before discussing how to fairly exchange bitcoins across two or more channels using Hashed Time-Locked Contracts. Finally, we highlighted challenges for route discovery in payment networks. It is our hope that this paper will motivate other researchers to begin tackling the open problems associated with Blockchain-based payment networks.

Acknowledgements. We thank Christian Decker, Joseph Poon and Rusty Russell for reviewing this paper. We also thank the participants of the lightning-dev mailing list for their insightful discussions. This work is supported in part by the European Research Council (ERC) Starting Grant (No. 306994) and the German Bundesministerium für Bildung und Forschung (BMBF) under grant agreement No. 13N13505.

References

1. Andresen, G.: BIP 101: increase maximum block size (2015). https://github.com/bitcoin/bips/blob/master/bip-0101.mediawiki Accessed 19 Apr 2016

[5] A wallet is frequently accessed and potentially connected to the internet.

2. Back, A., Corallo, M., Dashjr, L., Friedenbach, M., Maxwell, G., Miller, A., Poelstra, A., Timón, J., Wuille, P.: Enabling blockchain innovations with pegged sidechains (2014)
3. bitcoinj. https://bitcoinj.github.io/. Accessed 19 Apr 2016
4. CJP: Routing on the lightning network? (2015). http://lists.linuxfoundation.org/pipermail/lightning-dev/2015-July/000031.html. Accessed 19 Apr 2016
5. Croman, K., Decker, C., Eyal, I., Gencer, A.E., Juels, A., Kosba, A., Miller, A., Saxena, P., Shi, E., Sirer, E.G., Song, D., Wattenhofer, R.: On scaling decentralized blockchains. In: 3rd Workshop on Bitcoin and Blockchain Research (2016)
6. Decker, C., Wattenhofer, R.: A fast and scalable payment network with bitcoin duplex micropayment channels. In: Pelc, A., Schwarzmann, A.A. (eds.) SSS 2015. LNCS, vol. 9212, pp. 3–18. Springer, Heidelberg (2015)
7. Eyal, I., Gencer, A.E., Sirer, E.G., van Renesse, R.: Bitcoin-NG: a scalable blockchain protocol. In: Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation, NSDI, Santa Clara, CA, USA, 16–18 March 2016
8. Garzik, J.: BIP 100: making decentralized economic policy (2015). http://gtf.org/garzik/bitcoin/BIP100-blocksizechangeproposal.pdf. Accessed 19 Apr 2016
9. Lewenberg, Y., Sompolinsky, Y., Zohar, A.: Inclusive block chain protocols. In: Böhme, R., Okamoto, T. (eds.) FC 2015. LNCS, vol. 8975, pp. 528–547. Springer, Heidelberg (2015)
10. Lombrozo, E., Lau, J., Wuille, P.: BIP 141: segregated witness (2015). https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki. Accessed 19 Apr 2016
11. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008)
12. Pair, S.: A simple, adaptive block size limit (2016). https://medium.com/@spair/a-simple-adaptive-block-size-limit-748f7cbcfb75. Accessed 19 Apr 2016
13. Poon, J.: Payment re-routing (2015). http://lists.linuxfoundation.org/pipermail/lightning-dev/2015-July/000018.html. Accessed 19 Apr 2016
14. Poon, J., Dryja, T.: The bitcoin lightning network: scalable off-chain instant payments (2016). https://lightning.network/lightning-network-paper.pdf. Accessed 19 Apr 2016
15. Russell, R.: Loop attack with onion routing (2015). http://lists.linuxfoundation.org/pipermail/lightning-dev/2015-August/000153.html. Accessed 19 Apr 2016
16. Russell, R.: Reaching The Ground With Lightning (draft 0.2) (2015). https://github.com/ElementsProject/lightning/blob/master/doc/deployable-lightning.pdf. Accessed 19 Apr 2016
17. Russell, R.: Routing on the lightning network? (2015). http://lists.linuxfoundation.org/pipermail/lightning-dev/2015-July/000019.html. Accessed 19 Apr 2016
18. Sompolinsky, Y., Zohar, A.: Secure high-rate transaction processing in bitcoin. In: Böhme, R., Okamoto, T. (eds.) FC 2015. LNCS, vol. 8975, pp. 507–527. Springer, Heidelberg (2015)
19. Spilman, J.: Anti DoS for tx replacement (2013). https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-April/002433.html. Accessed 19 Apr 2016
20. Todd, P.: OP_ CHECKLOCKTIMEVERIFY (2014). https://github.com/bitcoin/bips/blob/master/bip-0065.mediawiki. Accessed 19 Apr 2016
21. Trillo, M.: Put to the stress test (2014). http://visatechmatters.tumblr.com/post/96025603185/put-to-the-stress-test-visanet-gets-pushed-to-the. Accessed 19 Apr 2016
22. Wuille, P.: BIP 66: strict DER signatures (2015). https://github.com/bitcoin/bips/blob/master/bip-0066.mediawiki. Accessed 19 Apr 2016