The Bitcoin Lightning Network

DRAFT Version 0.5

Joseph Poon <joseph@lightning.network>, Thaddeus Dryja <rx@awsomnet.org>

Abstract. The bitcoin protocol[1] can encompass the global financial transaction volume in all electronic payment systems today, without a single custodial 3rd party holding funds or requiring participants to have any more than a computer on a home broadband connection. A decentralized system is proposed whereby transactions are sent over a network of micropayment channels (a.k.a. payment channels or transaction channels) whose transfer of value occurs off-blockchain. If Bitcoin transactions can be signed with a new sighash type which addresses malleability, these transfers may occur between untrusted parties along the transfer route by contracts which are enforceable via broadcast over the bitcoin blockchain in the event of uncooperative or hostile participants, through a series of decrementing timelocks.

1. The Bitcoin Blockchain Scalability Problem

The blockchain as a payment platform, by itself, cannot cover the world’s commerce anytime in the near future. If each node in the bitcoin network must know about every single transaction that occurs globally, that may create a significant drag on the ability for the network to encompass all global financial transactions.

The payment network Visa is believed to do 45,000 peak transactions per second on its network during holidays, and hundreds of millions average per day. Currently, Bitcoin supports around 7 transactions per second with a 1 megabyte block limit. If we use an average of 300 bytes per bitcoin transaction and assume unlimited block sizes, an equivalent capacity to peak Visa transaction volume of 45,000/tps would be nearly 8 gigabytes per Bitcoin block, every ten minutes on average. Continuously, that would be over 400 petabytes per year. Clearly, that isn’t feasible today. Today’s personal computers cannot operate with that kind of bandwidth and storage. If Bitcoin is to replace all electronic payments in the future, not just Visa, as currently implemented it can only achieve a small portion of that, or at best, scale with extreme centralization of a few capital-intensive Bitcoin nodes and miners.

While it is possible that Moore’s Law will continue indefinitely, and the computational capacity for nodes to cost-effectively compute multi-gigabyte blocks may exist in the future, it is not a certainty. Relying on optimistic extrapolations of Moore's law isn't a plan, but a hope that one won't be needed.

To achieve much higher than 45,000 transactions per second using Bitcoin requires conducting transactions off the Bitcoin blockchain itself. It would be even better if the bitcoin network supported a near-unlimited number of transactions per second with extremely low fees for micropayments. Many micropayments can be sent sequentially between two parties to enable any size of payments.

“If a tree falls in the forest and no one is around to hear it, does it make a sound?”

Whether tree falling makes a sound questions the concept of relevant information affecting the wider universe, if nobody is around to hear the tree falling, then whether it made a sound or not is of no consequence. Similarly, in the blockchain, if only two participants care about a transaction, it’s not necessary for all other nodes in the bitcoin network to know about that transaction. It is instead preferable to only have the bare minimum of information on the blockchain. It is desirable for two individuals to net out their relationship at a later date, rather than detailing every transaction on the blockchain. This can be achieved by using time locks as a component to global consensus. As a result, Bitcoin can scale to billions of transactions per day with the computational power available today on a modern desktop computer.

2. A Network of Micropayment Channels Can Solve Scalability

Currently, bitcoin denominated micropayments are implemented by offloading the transactions to a custodian, whereby users are trusting 3rd party custodians to hold coins, update user balances, and allow deposits and withdrawals. Trusting 3rd parties to hold all of one’s funds creates counterparty risk; the consequences of such risk have been well-publicized.

Trusting 3rd parties to hold all of one’s funds creates counterparty risk and transaction costs. While currently many bitcoin services use this custodial model, it is possible to use native bitcoin transactions to scale to billions of users. We propose a system using bitcoin transaction scripting and micropayment channels, without custodial risk of theft. Sending many payments inside these micropayment channel enables one to send large amounts of funds to another party in a decentralized manner.

Micropayment channels[2][3] create a relationship between two parties to perpetually update balances, deferring the transaction broadcast to the blockchain in a single transaction netting out the total balance between those two parties. This permits the financial relationships between two parties to be trustlessly deferred to a later date, without risk of counterparty default. Micropayment channels use real bitcoin transactions, only electing to defer the broadcast to the blockchain in such a way whereby both parties can guarantee their current balance on the blockchain; this is not a trusted overlay network, payments in micropayment channels are real bitcoin communicated and exchanged off-chain.

However, micropayment channels only create a relationship between two parties. Requiring everyone to create channels with everyone else does little to solve the scalability problem. General Bitcoin scalability can be achieved using a large network of micropayment channels.

If we presume a large network of channels, and all Bitcoin users are participating on this graph by having at least one channel open on the Bitcoin blockchain, it is possible to create a near-infinite amount of transactions inside this network. The only transactions that are broadcast on the Bitcoin blockchain are channels closing due to uncooperative counterparties.

By encumbering the Bitcoin transaction outputs with a “hashlock” and “timelock”, the channel counterparty will be unable to outright steal funds and bitcoins can be exchanged without outright counterparty theft.

3. Hashlocked Bidirectional Micropayment Channels

Micropayment channels permit a simple deferral of a transaction state to be broadcast at a later time. The contracts are enforced by creating a responsibility for one party to broadcast transactions before or after certain dates. If the blockchain is a decentralized timestamping system, it is possible to use clocks as a component of decentralized consensus to determine data validity[4]. By creating timeframes where certain states can be broadcast and later invalidated, it is possible to create complex contracts using bitcoin transaction scripts.

There has been prior work on Hub-and-Spoke Micropayment Channels[5][6][7][8]. Implementing our hub-and-spoke network requires the malleability soft-fork described in Appendix B. This would enable near-infinite scalability while mitigating risks of intermediate node default. The implementation in Appendix A describes a method without requiring a soft-fork and may be used on the Bitcoin blockchain today with some minimal risk.

By chaining together multiple micropayment channels, it is possible to create a network of transaction paths. Paths can be routed using a BGP-like system, or the sender may designate a particular path to the recipient. The output scripts are encumbered by a hash, which is generated by the recipient. By disclosing the preimage of that hash, the recipient's counterparty will be able to pull funds along the route.

3.1. Hashed Timelock Contract (HTLC)

With a network of payment channels, a blockchain enforced contract must be constructed to prevent sender, recipient, and intermediaries from delaying or stealing funds.

An HTLC is opened by creating a transaction output which only the final recipient can redeem. The recipient first generates random data R, and hashes R using hash(R) to produce H. This information is provided directly from receiver to sender of funds, along with the recipient's bitcoin address. The sender routes their payment to the receiver. When the recipient has received an updated transaction in a micropayment channel, the recipient may elect to redeem the transaction by disclosing the random data R, which will ultimately pull funds from the sender.

The purpose of a Hashed Timelock Conctact is to require a message, R, to be known and disclosed in order for the transaction to be broadcast on the blockchain before a certain date.

In effect, Alice and Bob agree to the following clearing and settlement contract:

- If Bob can produce to Alice an unknown 20-byte random input data R from a known H, within three days, then Alice will settle the contract by paying Bob 0.1 BTC.
- If three days have elapsed, then the above clause is null and void and the clearing process is invalidated, both parties must not attempt to settle and claim payment after three days.
- Either party may (and should) pay out according to the terms of this contract in any method of the participants choosing and close out this contract early so long as both participants in this contract agree.
- Violation of the above terms will incur a maximum penalty of the funds locked up in this contract, to be paid to the non-violating counterparty.

The contract is created using this output script:

```
OP_DEPTH 3 OP_EQUAL OP_IF OP_HASH160 <hash160(R)> OP_EQUALVERIFY
OP_0 2 <AlicePubkey1> <BobPubkey1> 2 OP_CHECKMULTISIG OP_ELSE OP_0
2 <AlicePubkey2> <BobPubkey2> 2 OP_CHECKMULTISIG OP_END
```

The output is encumbered by a hash if the parties' first signature is provided, but not if both parties' second signatures are provided. Two sets of keys from both participants are used to enforce the ability to broadcast the spending transaction depending on whether R from hash (R) is known.

In effect, the goal of this output is to create two paths for the output: Path 1, if hash(R) is produced within three days, and Path 2. The two paths are mutually exclusive as they are the same output. Enforcement of whether the second path can be broadcast is done via a locktimed transaction. It is therefore possible to create the defined clearing and settlement contract by generating spends from this output script first.

The first path is the Settlement Transaction (S) which has a 2-of-2 output. The Settlement Transaction is consumed by a Settlement Delivery Transaction (SD) which pays out to the recipient in the event the input R is produced within the agreed upon contractual date. The Settlement Delivery transaction has a longer locktime, e.g. 40 days. This extra step is necessary to invalidate the Settlement Transaction. Per our example, only Alice sends the signature for S and SD to Bob.

The second path is the Timeout Transaction (T), consuming the same txout as the Settlement Transaction (S). The Timeout Transaction has a 2-of-2 output and is consumed by the Timeout Refund (TR) with a refund back to the original party and cancels the clearing process. The extra timelocked step from a Timeout Transaction to a Timeout Refund Transaction is necessary to invalidate the Timeout Transaction. Per our example, only Bob sends the signature for T and TR to Alice.

By having this extra step for the TR and SD transaction, one is able to cancel the contract itself by creating another transaction with no locktime giving the other party the full payout spending from S or T. This enforces a future contract term of requiring the counterparty to never broadcast a deprecated HTLC.

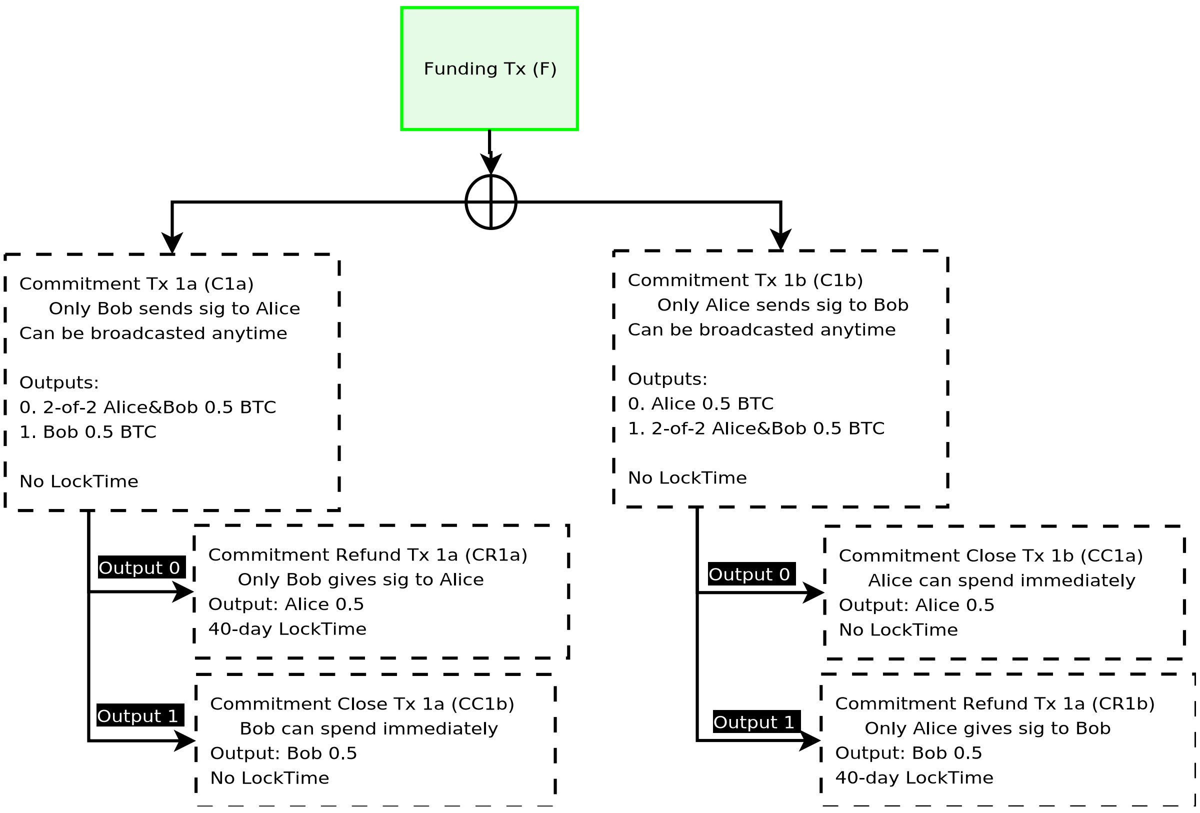
Transfers below the minimum IsStandard outptut size can be made with two payment outputs in the HTLC (with one output going to each party), with the net difference between the two outputs being the actual payment. Transactions can theoretically be sent as small as one Satoshi, however it may be possible that the fees paid to the intermediary may be higher.

This transaction, and spends dependent upon this transaction, function as a credible contract backed by enforcible action. The social agreement is to act honestly between participants on the channel, however if one party is non-cooperative, this transaction may be broadcast on the bitcoin blockchain to enforce the contract without the need for the counterparty to cooperate. In effect, the terms of the contract are programmed into the transactions itself. Which transactions get broadcast are dependent upon time and information disclosure.

3.2. Creating the Channel

In order to participate in this payment network, one must create a micropayment channel with another participant on the network. A Funding Transaction is created by either one or both of the channel participants. The output for this Funding Transaction is a single 2-of-2 multisig address controlled by both participants in the channel, henceforth named Alice and Bob. Both participants do not exchange signatures for the Funding Transaction until they have created spends from this 2-of-2 output refunding the original amount back to its respective funders.

A SIGHASH_NORMALIZED transaction is used to spend from this 2-of-2 Funding Transaction output, as normalized TXIDs are necessary to spend from a transaction which the signatures are not yet exchanged. SIGHASH_NORMALIZED and SIGHASH_NOINPUT, implemented using a soft-fork ensures transactions can be spent from before it is signed by all parties, as transactions would need to be signed to get a transaction ID without new sighash flags. See Appendix B for more information and implementation.



Funding Tx (F)
Commitment Tx 1a (C1a)
Only Bob sends sig to Alice
Can be broadcasted anytime
Outputs:
0. 2-of-2 Alice&Bob 0.5 BTC
1. Bob 0.5 BTC
No LockTime
Commitment Tx 1b (C1b)
Only Alice sends sig to Bob
Can be broadcasted anytime
Outputs:
0. Alice 0.5 BTC
1. 2-of-2 Alice&Bob 0.5 BTC
No LockTime
Output 0
Commitment Refund Tx 1a (CR1a)
Only Bob gives sig to Alice
Output: Alice 0.5
40-day LockTime
Output 1
Commitment Close Tx 1a (CC1b)
Bob can spend immediately
Output: Bob 0.5
No LockTime
Output 0
Commitment Close Tx 1b (CC1a)
Alice can spend immediately
Output: Alice 0.5
No LockTime
Output 1
Commitment Refund Tx 1a (CR1b)
Only Alice gives sig to Bob
Output: Bob 0.5
40-day LockTime

Fig 1. *Alice and Bob fund a transaction. CR1a, CR1b, C1a, and C1b signatures are shared as described, and F is exchanged and broadcast on the blockchain. Dashed lines denote a transaction which is not broadcast on the blockchain. Shaded-green/solid-line is a transaction which gets broadcast on the blockchain. In this paper, times such as 40 days are used for clarity, but it is preferable to use block height due to blockchain reorganization security.*

All spends from the Funding Transaction output, termed Commitment Transactions, consist of two half-signed transactions: one which Alice signs and gives to Bob (C1b), and another which Bob signs and gives to Alice (C1a). These two Commitment Transactions spend from the same output, and have different contents; only one can be broadcast on the blockchain. Either party may broadcast the Commitment Transaction by signing their received version and including the counterparty's signature.

The first two outputs from all Commitment Transactions contain a refund of the present balance between the counterparties. For new transactions, it should just be a refund to one or both participants, identical in amount to the Funding Transaction. The counterparty may immediately redeem the funds if the Funding Transaction, as well as the Commitment Transaction (which spends from the Funding Transaction) is included in the blockchain. The party which broadcast the transaction must wait until a pre-agreed time before they may redeem their transaction which is enforced by a 2-of-2 output -- for our examples, we will presume a 40-day locktime. This locktime is a 2-of-2 spend, a Commitment Close Transaction, from the Commitment Transaction. For instance, if Alice broadcasts the Funding Transaction and the Commitment Transaction, Bob will be able to receive a refund for whatever he put into the funding transaction using the Commitment Refund Transaction (CR1a), while Alice must wait 40 days before broadcasting her Commitment Close Transaction (CC1a) which includes an output to Alice. Bob does not have to wait because

Alice was the party who broadcast the Commitment Transaction, so Alice is attesting that Bob should receive at least a certain amount, which is automatically sent to Bob, whereas some timelocked proof is necessary to show Alice’s refund is correct.

When initially funding a channel, parties exchange their signatures for the Commitment Transaction which refunds both parties their respective contributions, and then the Funding Transaction signatures are exchanged. In the event the counterparty fails to exchange the Funding Transaction signatures after the Commitment Signatures are exchanged, one must double-spend their respective inputs to the Funding Transaction.

Note that either party may broadcast their version of the Commitment Transaction, and only one may be broadcast. Parties are only able to broadcast the version that they have received. When correctly broadcasting one’s version of the Commitment Transaction, one must wait until the agreed upon locktime before the funds are redeemable.

Outputs from each Commitment Transaction use a unique set of addresses. For example, Bob sets all his outputs from Commitment Transactions to use BobKeyB and this key is only for C1b, whereas Bob uses BobKeyA for C1a. Note that all spends from the Funding Transaction to use as the input for the Commitment Transaction still reuse the same key, as it is not possible to change a transaction that is included in the blockchain. This dual-key structure is necessary for future invalidation of transactions.

Once the funding transaction is committed to the blockchain, the channel is open for further transactions.

3.3. Transaction Clearing: Making a Payment in a Channel

To send funds from one party to another in the channel, one creates a Hashed Timelock Contract (HTLC), as described previously, to clear a payment off-chain.

The funds from the first two outputs of the most recent Commitment Transaction are deducted and a new pair of Commitment Transactions are created. New HTLC outputs created in the new pair of Commitment Transactions, whose output amount is the amount one wishes to send (including fees).

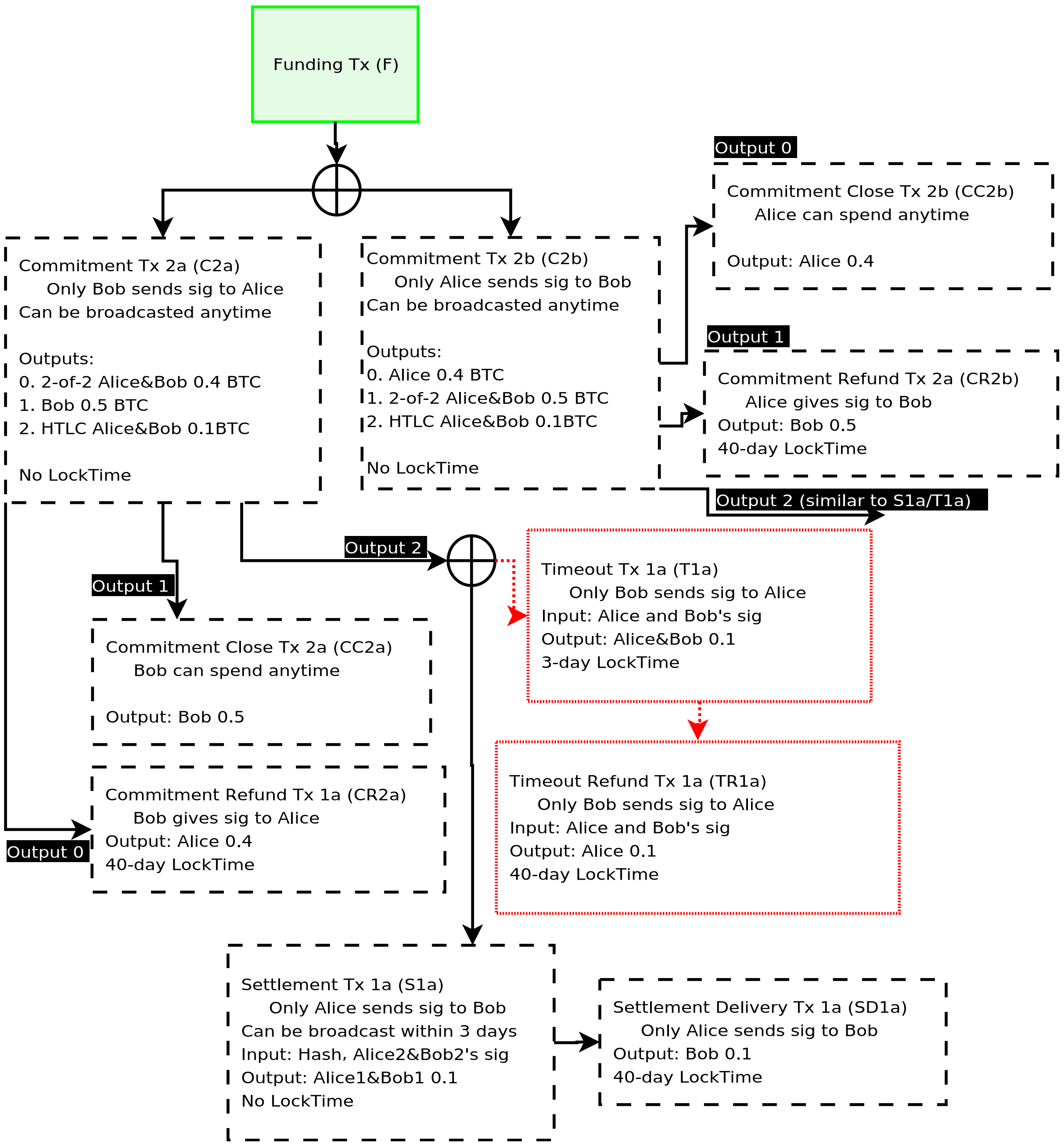

Funding Tx (F)
Commitment Tx 2a (C2a)
Only Bob sends sig to Alice
Can be broadcasted anytime
Outputs:
0. 2-of-2 Alice&Bob 0.4 BTC
1. Bob 0.5 BTC
2. HTLC Alice&Bob 0.1BTC
No LockTime
Commitment Tx 2b (C2b)
Only Alice sends sig to Bob
Can be broadcasted anytime
Outputs:
0. Alice 0.4 BTC
1. 2-of-2 Alice&Bob 0.5 BTC
2. HTLC Alice&Bob 0.1BTC
No LockTime
Output 0
Commitment Close Tx 2b (CC2b)
Alice can spend anytime
Output: Alice 0.4
Output 1
Commitment Refund Tx 2a (CR2b)
Alice gives sig to Bob
Output: Bob 0.5
40-day LockTime
Output 2 (similar to S1a/T1a)
Output 2
Timeout Tx 1a (T1a)
Only Bob sends sig to Alice
Input: Alice and Bob's sig
Output: Alice&Bob 0.1
3-day LockTime
Output 1
Commitment Close Tx 2a (CC2a)
Bob can spend anytime
Output: Bob 0.5
Timeout Refund Tx 1a (TR1a)
Only Bob sends sig to Alice
Input: Alice and Bob's sig
Output: Alice 0.1
40-day LockTime
Commitment Refund Tx 1a (CR2a)
Bob gives sig to Alice
Output: Alice 0.4
40-day LockTime
Output 0
Settlement Tx 1a (S1a)
Only Alice sends sig to Bob
Can be broadcast within 3 days
Input: Hash, Alice2&Bob2's sig
Output: Alice1&Bob1 0.1
No LockTime
Settlement Delivery Tx 1a (SD1a)
Only Alice sends sig to Bob
Output: Bob 0.1
40-day LockTime

Fig 2. *Alice and Bob update their Commitment Transactions and exchange signatures, after the necessary signatures are shared for the transactions spending from the Commitment Transaction. Red/thin-dashed-lines imply a transaction which only gets broadcast when something goes wrong. Note that the Commitment Transaction uses the number 2 and the HTLC uses the number 1, as it is the first HTLC. While it may appear that this is too complicated and creates too many transactions, these transactions only get broadcast if the counterparty is uncooperative, in nearly every instance both parties net out their balances and do a single transaction spending from the Funding Transaction when they wish to close out the channel.*

It is presumed that all HTLC spends, specifically, the Settlement Tx and the Timeout Tx are SIGHASH_NOINPUT, which can be spent from any transaction. Note that every HTLC output from a Commitment Transaction must use a unique pubkey and address. Reusing pubkeys will likely result in coin theft. It is possible to use this with SIGHASH_NORMALIZED, but it may require more signatures since every new Commitment Transaction must re-sign all pending HTLC spends and the transactions which spend from those HTLC spends.

Note that each party exchanges four new sets of pubkeys/addresses for the new HTLCs (As there are two necessary for the HTLC output, see the HTLC section for AliceKey1 and AliceKey2), which are spent in S1a, SD1a, T1a, TR1a or S1b, SD1b, T1b, TR1b (depending upon which Commitment Transaction is broadcast).

Additionally, two new sets of pubkeys/addresses are used for the two new Commitment Transactions, which get redeemed in the Commitment Refund Transactions.

In total six keys are created to make a new HTLC contract in this channel.

It is possible that new HTLCs will be appended in the future, while old ones are still in use for new Commitment Transactions. For this reason, each party uses creates new keys.

The Commitment Transactions will only be exchanged when all parties signed transactions spending their outputs. Presume Alice wishes to clear a payment to/through Bob. The steps to clear a payment are as follows:

1. Alice signs and sends Bob her signature for: CR2b, S1, SD1.
2. Bob signs and sends Alice his signature for: CR2a, T1, TR1, C2a. At this point, Alice is able to broadcast C2a or C1a.
3. Alice signs and sends Bob her signature for C2b. At this point Bob is able to broadcast C2b or C1b. Alice may broadcast C2a and the balances will be the same. She gives Bob her **private key** from the Commitment Close Transaction from C1a. By giving her private key for C1a, Alice repudiates transaction C1a, assuring Bob that she will not broadcast it.
4. Bob similarly gives Alice his **private key** for C1b. Both parties will only broadcast C2a or C2b. The transaction is now cleared. Bob is clear to take action to settle the HTLC.

This four-step clearing process ensures that neither party can steal the funds of the transaction. When Step 2 has completed, Alice is able to broadcast either the C1a or C2a series of transactions. Bob doesn't care which one Alice broadcasts, since the balances will remain the same, presuming the hashlock is not redeemable. Alice then gives Bob sufficient data to broadcast either C1a or C2a, where Alice similarly doesn't care which is used. They then invalidate C1a and C1b by exchanging the private keys to spend the Commitment Close Transactions.

If these four steps are completed, and Alice broadcasts C1a, Bob will have the private key to redeem CR1a for himself, as he has Alice's C1a key and he has not shared his CR1a key -- he only shared his CR1b key. Since Alice's version of CR1a has a locktime of 40 days, Bob can immediately broadcast his transaction taking all of Alice's coins well before the 40 day locktime. Thus, Alice will not broadcast any previous Commitment Transactions, such as C1a, and correspondingly Bob will not broadcast C1b. If they did broadcast such a spend, their counterparty would take all the money in the channel. Both parties are motivated to delete their old copies of Commitment Transactions.

Also, note that only Alice can broadcast C1a, and only Bob can broadcast C1b -- this has not changed.

When one party wishes to append another HTLC, the same process occurs. All previous HTLCs outputs in the Commitment Transaction are kept in the new Commitment Transaction.

3.4. HTLC Settlement

After an HTLC has been established, it may become desirable to settle the cleared transaction. If it is within the agreed time in the contract, the contract can be net settled by removing the HTLC and adding the balance to the output in the Commitment Transaction to be spent by the Commitment Refund or Commitment Close Transaction. The steps to do so are the same as adding an HTLC. A new Commitment Transaction superseding the old one is established via novation (contract/transaction replacement), except an HTLC is removed instead of added.

Additionally, HTLC private keys are disclosed to the counterparty to enforce non-broadcast of the older Commitment Transactions in step 3 and 4. Alice discloses to Bob her C2a keys, and Bob discloses to Alice his C2b keys. That way if Alice broadcasts C2a, Bob is able to make any spends necessary to give himself all funds within the HTLC. Alice will also be able to take all funds from the HTLC if Bob broadcasts C2b or any older Commitment Transaction containing the HTLC. Alice does NOT broadcast her private key used in the counterparty's half of the Commitment Transaction (e.g. C2b), similarly Bob does not do the same for Alice's half.

As a result, when the HTLC gets settled in a new Commitment Transaction pair (C3a and C3b) with no HTLC output, and the balances get settled in CC3 and CR3, if C2a gets broadcast, Bob will be able to spend from the HTLC output in C2a. Bob will also be able to spend from S1a or T1a because the signed spends SD1 and TR1 are locked for 40 days. Since Bob can create a new spend from S1a or T1a to send the coins to himself, the SD1a and TR1a payout to Alice are effectively invalidated. Therefore, Alice will not be able to steal coins from an already settled HTLC by broadcasting old transactions.

Successful settlement is defined by the recipient being able to prove that they have R from hash(R) defined in the HTLC. By producing the input to the hash, both parties know that the transaction may be redeemed by broadcasting the transactions on the blockchain. The parties may then cooperate and settle the transaction with a new Commitment Transaction (and invalidating the previous Commitment Transaction).

If the current date is after the agreed upon timeout, both parties should update with a new Commitment Transaction refunding the balance and removing the HTLC, as the clearing process has failed.

In nearly all situations, all participants want to close out the transaction as early as possible instead of waiting until expiry to keep their channels open.

3.5. Closing Out a Channel

In the event the counterparty does not wish to settle using a new Commitment Transaction or is otherwise uncooperative, a delivery event occurs and all current transactions must be broadcast to the blockchain, closing the channel. The Funding Transaction, most recent Commitment Transactions, and all pending HTLCs are broadcast to the blockchain.

One must be able to enforce the contract defined in the HTLC by settling the contract with ones' counterparty. If one is unable to do so, then one must broadcast to the blockchain before a timeout is enforced if one is able to settle the HTLC. If timeout has occurred and the failure cannot be closed with a new Commitment Transaction, a broadcast on the blockchain must also occur.

In the above HTLC, Bob **must** settle the transaction within 3 days. In the event Alice cannot be contacted, Bob **must** broadcast the Commitment Transaction and HTLC Timeout Transaction on day 3. If Bob does not, then the contract timeout terms may not be enforceable.

The result is a system whereby contract enforcement occurs on the blockchain according to the rules defined in the contract. Clearing states are defined by timelocks, and the ability to produce the preimage R of hash(R).

Both parties are able to send as many payments to their counterparty as they wish, as long as they have funds available in the channel, knowing that in the event of disagreements they can broadcast to the blockchain the current state at any time.

In nearly every instance, the outputs from the Funding Transaction *will never be broadcast to the blockchain.* They exist as a failsafe in case the other party is non-cooperative, much like how most legal contracts do not lead to lawsuits and judgments. A proven contract enforcement mechanism is sufficient incentive for both parties to cooperate.

When either party wishes to close out a channel cooperatively, they will be able to do so via novation by contacting the other party and spending from the Funding Transaction with an output of the most current Commitment Transaction. No further payments may occur in the channel.

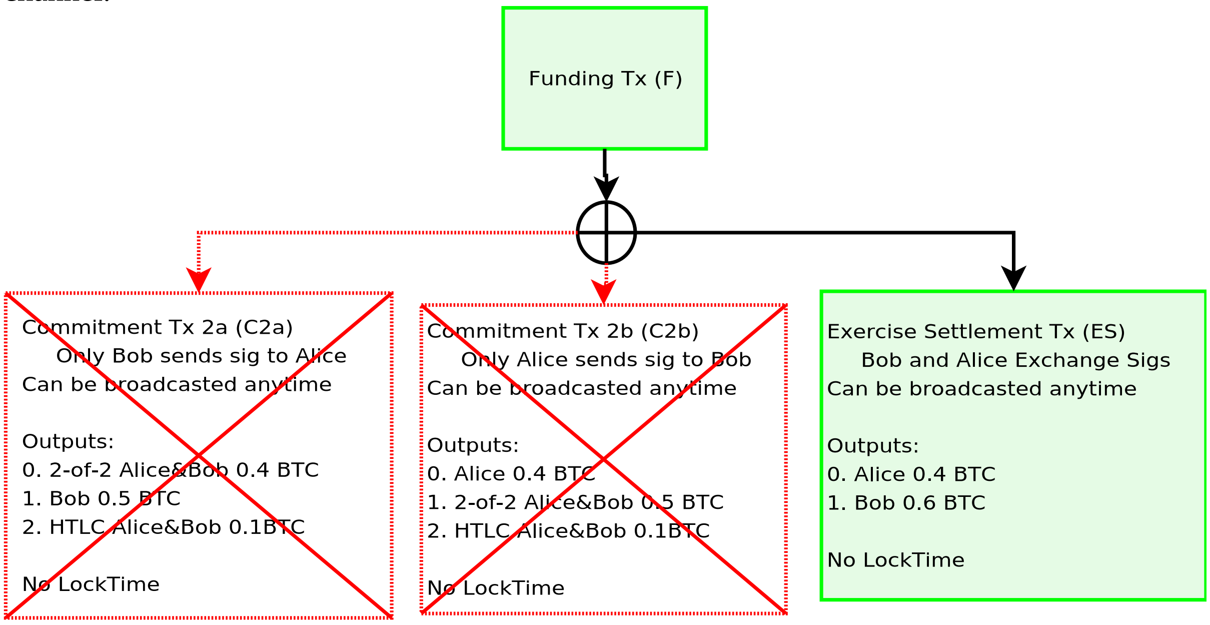

Funding Tx (F)
Commitment Tx 2a (C2a)
Only Bob sends sig to Alice
Can be broadcasted anytime
Outputs:
0. 2-of-2 Alice&Bob 0.4 BTC
1. Bob 0.5 BTC
2. HTLC Alice&Bob 0.1BTC
No LockTime
Commitment Tx 2b (C2b)
Only Alice sends sig to Bob
Can be broadcasted anytime
Outputs:
0. Alice 0.4 BTC
1. 2-of-2 Alice&Bob 0.5 BTC
2. HTLC Alice&Bob 0.1BTC
No LockTime
Exercise Settlement Tx (ES)
Bob and Alice Exchange Sigs
Can be broadcasted anytime
Outputs:
0. Alice 0.4 BTC
1. Bob 0.6 BTC
No LockTime

Fig 3. *Both parties take the balances in the current Commitment Transaction and spend from the Funding Transaction with a Exercise Settlement Transaction (ES). If the most recent Commitment Transaction gets broadcast instead, the payout (less fees) will be the same.*

Channels must close out before the earliest locktime defined in Commitment Refund (CR), Settlement Delivery (SD), or Timeout Refund (TR) transactions. Failure to close the channel via Exercise Settlement or broadcast of the most recent transaction tree (in the event of an uncooperative counterparty) before the most recent CR, SD, or TR locktime may result in loss of funds.

3.6. Key Storage

Keys are generated using BIP 0032 Hierarchical Deterministic Wallets[9]. Keys are pre-generated by both parties. Keys are generated in a merkle-tree and very deep. For instance, Alice pre-generates one million keys, each key being a child of the previous key. Alice allocates according to some deterministic manner which keys to use. For example, she starts with the child deepest in the tree to generate many sub-keys for day 1. This key is used as a master key for all keys generated on day 1. She gives Bob the address she wishes to use for the next transaction, and discloses the private key to Bob when it becomes invalidated. When Alice discloses to Bob all private keys derived from the day 1 master key and does not wish to continue using that master key, she can disclose the day 1 master key to Bob. At this point, Bob does not need to store all the keys derived from the day 1 master key. Bob does the same for Alice and gives her his day 1 key.

When all Day 2 private keys have been exchanged, for example by day 5, Alice discloses her Day 2 key. Bob is able to generate the Day 1 key from the Day 2 key, as the Day 1 key is a child of the Day 2 key as well.

If a counterparty broadcasts the wrong Commitment Transaction, which private key to use in a transaction to recover funds can either be brute forced, or if both parties agree, they can use the sequence id number when creating the transaction to identify which sets of keys are used.

This enables participants in a channel to have prior output states (transactions) invalidated by both parties without using much data at all. By disclosing private keys pre-arranged in a merkle-tree, it is possible to invalidate millions of old transactions with only a few kilobytes of data per channel. Core channels in the Lightning Network can conduct billions of transactions without a need for significant storage costs.

3.7. Blockchain Transaction Fees

It is possible for each participant to generate different versions of transactions to ascribe blame to who broadcast the transaction on the blockchain. By knowing who broadcast a transactions and ascribing blame, a 3rd party service can be used to hold fees in a 2-of-3 multisig escrow. If one wishes to broadcast the transaction chain instead of agreeing to do a Funding Close or replacement with a new Commitment Transaction, one would communicate with the 3rd party and broadcast it to the blockchain. If the counterparty refuses the notice from the 3rd party to cooperate, the penalty is rewarded to the non-cooperative party. In most instances, participants may be indifferent to the transaction fees in the event of an uncooperative counterparty.

Note that non-cooperation is only a relevant factor to one’s immediate counterparty in the micropayment channel, and no other parties. One should pick counterparties in the channel who will be cooperative, so a 3rd party service is not required. Note that this does not require trust among the rest of the network, and is only relevant for the comparatively minor transaction fees. The less trusted party may just be the one responsible for transaction fees.

4. Pay to Contract

It is possible construct a cryptographically provable “Delivery Versus Payment” contract, or pay-to-contract[10] , as proof of payment. This proof can be established as knowledge of the input R from hash(R) as payment of a certain value. By embedding the contract between the buyer and seller that knowing R is proof of funds sent, the recipient of funds has no incentive to disclose R unless they have certainty that they will receive payment. When the funds eventually get pulled from the buyer by their counterparty in their micropayment channel, R is disclosed as part of that pull of funds. One can design paper legal documents that specify knowledge or disclosure of R implies fulfillment of payment. The sender can arrange a cryptographically signed contract for knowledge of inputs for hashes as fulfillment of contracts before payment.

5. The Bitcoin Lightning Network

By having a micropayment channel with contracts encumbered by hashlocks and timelocks, it is possible to clear transactions over a multi-hop payment network using a series of decrementing timelocks without additional central clearinghouses.

Traditionally, financial markets clear transactions by transferring the obligation for delivery at a central point and settle by transferring ownership through this central hub. Bank wire and fund transfer systems (such as ACH and the Visa card network), or equities clearinghouses (such as the DTCC) operate in this manner.

As Bitcoin enables programmatic money, it is possible to create transactions without contacting a central clearinghouse. Transactions can execute off-chain with no 3rd party which collects all funds before disbursing it -- only transactions with uncooperative channel counterparties become automatically adjudicated on the blockchain.

The obligation to deliver funds to an end-recipient is achieved through a process of chained delegation. Each participant along the path assumes the obligation to deliver to a particular recipient. They pass on this obligation to the next participant in the path. The obligation of each subsequent participant along the path, defined in their respective HTLCs, has a shorter time to completion compared to the prior participant. This way each participant is sure that they will be able to claim funds when the obligation is sent along the path.

Bitcoin Transaction Scripting, a form of what some call an implementation of "Smart Contracts"[11], enables systems without trusted custodial clearinghouses or escrow services.

5.1. Decrementing Timelocks

Presume Alice wishes to send 0.001 BTC to Dave. She locates a route through Bob and Carol. The transfer path would be Alice to Bob to Carol to Dave.

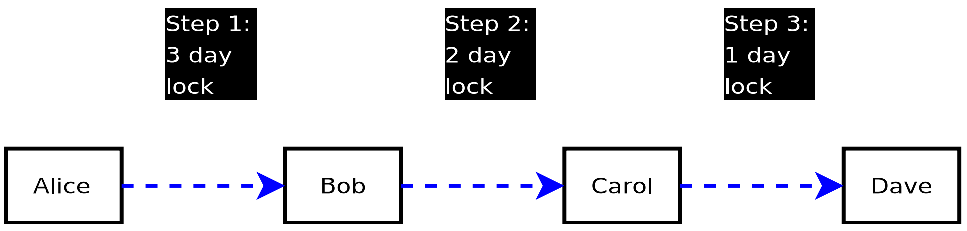
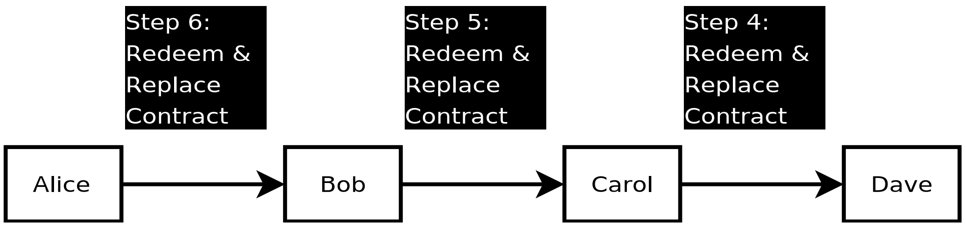


Fig 4. *Payment over the Lightning Network using HTLCs.*

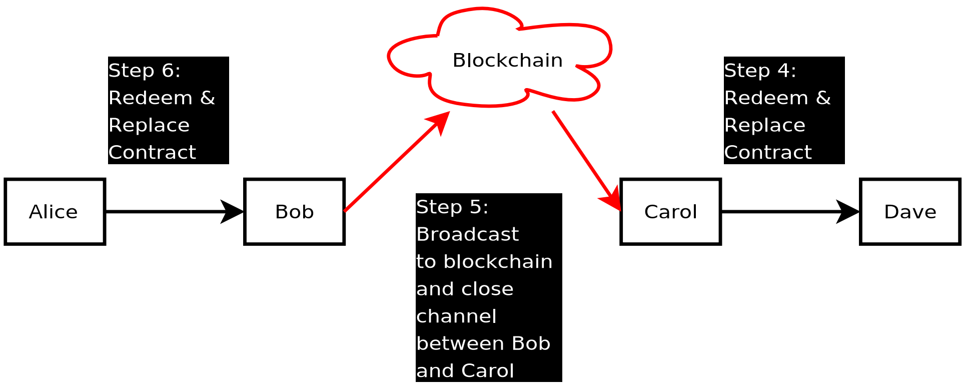
When Alice sends payment to Dave through Bob and Carol, she requests from Dave hash(R) to use for this payment. Alice then counts the amount of hops until the recipient and uses that as the HTLC expiry. In this case, she sets the HTLC expiry at 3 days. Bob then creates an HTLC with Carol with an expiry of 2 days, and Carol does the same with Dave with an expiry of 1 day. Dave is now free to disclose R to Carol, and both parties will likely agree to immediate settlement via novation with a replacement Commitment Transaction. This then occurs step-by-step back to Alice. Note that this occurs off-chain, and nothing is broadcast to the blockchain when all parties are cooperative.


Step 6: Redeem & Replace Contract
Step 5: Redeem & Replace Contract
Step 4: Redeem & Replace Contract
Alice
Bob
Carol
Dave
Fig 5. *Settlement of HTLC, Alice's funds get sent to Dave.*

Decrementing timelocks are used so that all parties along the path know that the disclosure of R will be able to pull funds, since they will at worst be pulling funds after the date whereby they must receive R. If Dave does not produce R within 1 day to Carol, then Carol will be able to close out the HTLC. If Dave broadcasts R after 1 day, then he will not be able to pull funds from Carol. Carol's responsibility to Bob occurs on day 2, so Carol will never be responsible for payment to Dave without an ability to pull funds from Bob provided that she updates her transaction with Dave to the blockchain or via novation.

In the event that R gets disclosed to the participants halfway through expiry along the path (e.g. day 2), then it is possible for some parties along the path to be enriched. The sender will be able to know R, so due to Pay to Contract, the payment will have been legally fulfilled even though the receiver did not receive the funds. Therefore, the receiver will never disclose R unless they are certain they will receive payment from one of their channel counterparties.

In the event a party outright disconnects, the counterparty will be responsible for broadcasting the current Commitment Transaction state in the channel to the blockchain. Only the failed non-responsive channel state gets closed out on the blockchain, all other channels should continue to update their Commitment Transactions via novation inside the channel. Therefore, counterparty risk for transaction fees are only exposed to direct channel counterparties. If a node along the path decides to become unresponsive, the participants not directly connected to that node suffer only decreased time-value of their funds by not conducting early settlement before the HTLC close.


Blockchain
Step 6: Redeem & Replace Contract
Step 4: Redeem & Replace Contract
Alice
Bob
Carol
Dave
Step 5: Broadcast to blockchain and close channel between Bob and Carol
Fig 6. *Only the non-responsive channels get broadcast on the blockchain, all others are settled off-chain via novation.*

5.2. Payment Amount

It is necessary to use a small payment per HTLC. One should not use an extremely high payment, in case the payment does not fully route to its destination. If the payment does not reach its destination and one of the participants along the path is uncooperative, it is possible that the sender must wait until the expiry before receiving a refund. Since transactions don't hit the blockchain with cooperative channel counterparties, it is recommended to use as small of a payment as possible. A tradeoff exists between locking up transaction fees on each hop versus the desire to use as small transaction fee as possible. Smaller transfers with more intermediaries imply a higher percentage paid as Lightning Network fees to the intermediaries.

5.3. Clearing Failure and Rerouting

If a transaction fails to reach its final destination, the receiver should send an equal payment to the sender with the same hash, but not disclose R. This will net out the disclosure of the hash for the sender, but may not for the receiver. The receiver, who generated the hash, should discard R and never broadcast it. If one channel along the path cannot be contacted, then the channels may elect to wait until the path expires, which all participants will likely close out the HTLC as unsettled without any payment with a new Commitment Transaction.

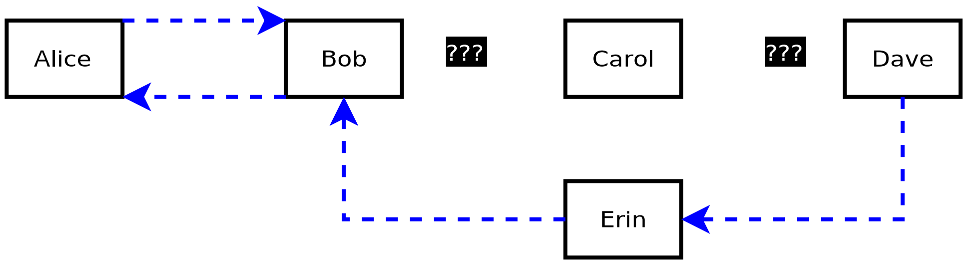


Fig 7. *Dave creates a path back to Alice after Alice fails to send funds to Dave, because Carol is uncooperative. The input R from hash(R) is never brodcast by Dave, because Carol did not complete her actions. If R was broadcast, Alice will break-even. Dave, who controls R should never broadcast R because he may not receive funds from Carol, he should let the contracts expire. Alice and Bob have the option to net out and close the contract early, as well, in this diagram.*

If the refund route is the same as the payment route, and there are no half-signed contracts whereby one party may be able to steal funds, it is possible to outright cancel the transaction by replacing it with a new Commitment Transaction starting with the most recent node who participated in the HTLC.

It is also possible to clear out a channel by creating an alternate route path in which payment will occur in the opposite direction (netting out to zero) and/or creating an entirely alternate route for the payment path. This will create a time-value of money for disclosing inputs to hashes on the Lightning Network. Participants may specialize in high connectivity between nodes and offering to offload contract hashlocks from other nodes for a fee. These participants will agree to payments which net out to zero (plus fees), but are loaning bitcoins for a set time period. Most likely, these entities with low demand for channel resources will be end-users who are already connected to multiple well-connected nodes. When an end-user connects to a node, the node may ask the client to lock up their funds for several days to another channel the client has established for a fee. This can be achieved by having the new transactions require a new hash(Y) from input Y in addition to the existing hash which may be generated by any participant, but must disclose Y only after a full circle is established. The new participant has the same responsibility as well as the same timelocks as the old participant being replaced. It is also possible that the one new participant replaces multiple hops.

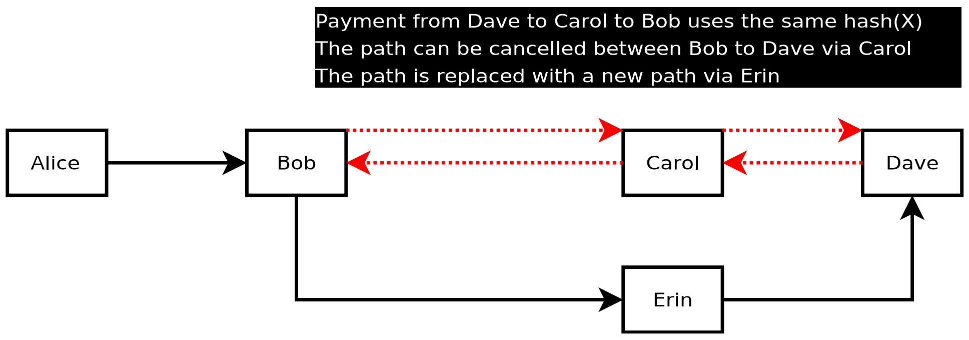

Payment from Dave to Carol to Bob uses the same hash(X)
The path can be cancelled between Bob to Dave via Carol
The path is replaced with a new path via Erin
Alice
Bob
Carol
Dave
Erin

Fig 8. *Erin is connected to both Bob and Dave. If Bob wishes to free up his channel with Carol, since that channel is active and very profitable, Bob can offload the payment to Dave via Erin. Since Erin has extra bitcoin available, she will be able to collect some fee for offloading the channel between Bob and Carol as well as between Carol and Dave. The channels between Bob and Carol as well as Carol and Dave are undone and no longer have the HTLC, nor has payment occurred on that path, it will occur on the path involving Erin. This is achieved by creating a new payment from Dave to Carol to Bob contingent upon Erin constructing an HTLC. The payment in dashed lines (red) are netted out to zero and settled via a new Commitment Contract.*

5.4. Payment Routing

It is theoretically possible to build a routing path implicitly from observing 2-of-2 multisigs on the blockchain to build a routing table. Note, however, this is not feasible with pay-to-script-hash transaction outputs, which can be resolved out-of-band from the bitcoin protocol via a 3rd party routing service. Building a routing table will become necessary for large operators, and is largely a solved problem (e.g. BGP, CJBDNS). Eventually, with optimizations, the network will look a lot like the correspondent banking network, or Tier-1 ISPs. Similar to how packets still reach their destination on your home network connection, not all participants need to have a full BGP routing table. The core Tier-1 routes can be online all the time -- while nodes at the edges, such as average users, would be connected intermittently.

5.5. Fees

Lightning Network Fees, which differ from blockchain fees, are paid directly between participants within the channel. The fees pay for the time-value of money for consuming the channel for a determined maximum period of time, and for counterparty risk of non-communication.

Counterparty risk for fees only exist with one’s direct channel counterparty. If a node two hops away decides to disconnect and their transaction gets broadcast on the blockchain, one’s direct counterparties should not broadcast on the blockchain, but continue to update via novation with a new Commitment Transaction. See the HTLC section on fees for more information about counterparty risk.

The time-value of fees pays for consuming time (e.g. 3 days) and is conceptually equivalent to a gold lease rate without custodial risk; it is the time-value for using up the access to money for a very short duration. Since certain paths may become very profitable in one direction, it is possible for fees to be negative to encourage the channel to be available for those profitable paths.

6. Risks

The primary risks relate to timelock expiration. Additionally, for core nodes and possibly some merchants, funds are held online ("hot wallet").

6.1. Improper Timelocks

Participants must choose timelocks with sufficient amount of time. If insufficient time is given, it is possible that transactions believed to be invalid will become valid, enabling coin theft by the counterparty. There is a trade-off between longer timelocks and the time value of money.

6.2. Forced Expiration Spam

Forced expiration of many transactions may be the greatest systemic risk when using the Lightning Network. If a malicious participant creates many channels and forces them all to expire at once, these may overwhelm block data capacity, forcing expiration and broadcast to the blockchain. The result would be a mass spam on the bitcoin network. The spam may delay transactions to the point where other locktimed transactions become valid.

This may be mitigated by permitting one transaction replacement on all pending transactions. Anti-spam can be used by permitting one transaction replacement of a higher sequence number by an opposite of an even or odd number. E.g. if an odd sequence number was broadcasted, permit a replacement to a higher even number only once. Transactions would use the sequence number in an orderly way to replace other transactions. This mitigates the risk assuming honest miners. There may be some further consensus mining optimizations, as well. This attack is extremely high risk, as incorrect broadcast of Commitment Transactions entail a full penalty of all funds in the channel.

Additionally, one may attempt to steal HTLC transactions by forcing a timeout transaction to go through when it should not. This can be easily mitigated by having each transfer inside the channel be lower than the total transaction fees used. Since transactions are extremely cheap and do not hit the blockchain with cooperative channel counterparties, large transfers of value can be split into many small transfers. This attempt can only work if the blocks are completely full for a long time. While it is possible to mitigate it using a longer HTLC timeout duration, a better method is to use a blocksize soft-cap.

If this type of transaction becomes the dominant form of transactions which are included on the blockchain, it may become necessary to increase the block size and run a soft-cap structure as described in the section below. Creating a bitcoin soft-cap can create sufficient penalties and disincentives to be highly unprofitable and unsuccessful for attackers, as attackers lose all their funds from broadcasting the wrong transaction, to the point where it will never occur.

6.3. Coin Theft via Hacking

As parties must be online and using private keys to sign, there is a possibility that if one's computer is compromised, that coins will be stolen by the counterparty. While there may be methods to mitigate the threat for the sender and the receiver, the intermediary nodes must be online and will likely be processing the transaction automatically. For this reason, the intermediary nodes will be at risk and should not be holding a substantial amount of money in this "hot wallet." Intermediary nodes which have better security will likely be able to out-compete others in the long run and be able to conduct greater transaction volume due to lower fees. Historically, one of the largest component of fees/interest is from various forms of counterparty risk -- in Bitcoin it is possible that the largest component in fees will be derived from security risk premiums.

A Funding Transaction may have multiple outputs with multiple Commitment Transactions, with the Funding Transaction and some Commitment Transactions stored offline. It is possible to create an equivalent to a “Checking Account” and “Savings Account” by moving funds between outputs from a Funding Transaction, with the “Savings Account” stored offline and requiring additional signatures from security services.

6.4. Data Loss

When one party loses data, it is possible for the counterparty to steal funds. This can be mitigated by having a 3rd party data storage service where encrypted data gets sent to this 3rd party service which the 3rd party cannot decrypt. Additionally, one should choose channel counterparties who are responsible and willing to provide the current state, with some periodic tests of honesty.

6.5. Forgetting to Broadcast the Transaction in Time

If one does not broadcast a transaction at the correct time, the counterparty may steal funds. This can be mitigated by having a designated 3rd party to send funds. An output fee can be added to create an incentive for this 3rd party to watch the network. Further, this can also be mitigated by implementing relative OP_CHECKLOCKTIMEVERIFY.

7. Relative OP_CHECKLOCKTIMEVERIFY: Solving long-term timelocks

It may be undesirable to lock up funds for a long periods of time. A tradeoff between transaction fees due to blockchain broadcasts and trusting the other party in the channel to cooperate and stay online exists.

A relative OP_CHECKLOCKTIMEVERIFY[12] can resolve this problem. By using a relative locktime opcode, one can invalidate transactions after they are broadcast independent of dates. It would therefore be possible to have transactions with very low 2-of-2 locktimes. The transactions are then sent to another 2-of-2 for a determinate period of time, for example 2 weeks. When that transaction is broadcast on the blockchain, the funds cannot be spent for two weeks, whose counter begins upon blockchain inclusion.

When an incorrect transaction gets broadcast with a two week relative OP_CHECKLOCKTIMEVERIFY, the counterparty is free to spend from that locked transaction for the next two weeks and take all funds as a penalty if the counterparty has the signatures or private keys. If one discloses to the counterparty this information, that transaction is in-effect invalidated, as the counterparty may spend from that 2-of-2 relative locktimed transaction within those two weeks. If the transaction is not redeemed after two weeks, then the part of the script checking for the relative OP_CHECKLOCKTIMEVERIFY would be valid.

Alternatively, is also possible to have something like an “OP_DEPTHLESSTHANVERIFY” to compare the input’s block depth with a number on the stack. If the input is in a block which is less than the specified number of blocks deep, the script continues. If the input is too deep, the transaction is marked as invalid. There is a minimum allowable depth comparison to prevent reorg risks.

8. Block Size Increases and Consensus

If we presume that a decentralized payment network exists and one person will make 3 blockchain transactions per year on average, Bitcoin will be able to support over 35 million users with 1MB blocks in ideal circumstances (assuming 2000 transactions per MB). This is quite limited, and an increase of the block size may be necessary to support everyone in the world using Bitcoin. Any increase of the block size is a hard fork, meaning all nodes will need to update their wallets if they wish to participate in the network with the larger blocks.

While it may appear as though this system will mitigate block size increases in the short term, it will increase the necessity of a block size increase in the long term. Creating a credible threat that spamming the blockchain to encourage transactions to timeout becomes imperative.

To mitigate timelock spam vulnerabilities, non-miner and miners' consensus rules may also differ if the miners' consensus rules are more restrictive. Non-miners may accept blocks over 1MB, while miners may have different soft-caps on block sizes. If a block size is above that cap, then that is viewed as an invalid block by other miners, but not by non-miners. The miners will only build the chain on blocks which are valid according to the agreed-upon soft-cap. This permits miners to agree on raising the block size limit without requiring frequent hard-forks from clients, so long as the amount raised by miners does not go over the clients' hard limit. This mitigates the risk of mass-expiry of transactions at once. All transactions which are not redeemed via Exercise Settlement (ES) may have a very high fee attached, and miners may use a consensus rule whereby those transactions do not apply to the soft-cap, making it very likely the correct transactions will enter the blockchain.

This can be achieved by rejecting blocks which violate agreed terms up to 1-3 block height, or by embedding some consensus rule as a vote in the coinbase transaction, looking back in some set number of blocks (e.g. past 1000 blocks).

When transactions are viewed as circuits and contracts instead of transaction packets, the consensus risks can be measured by the amount of time available to cover the UTXO set controlled by hostile parties. In effect, the upper bound of the UTXO size is determined by transaction fees and the standard minimum transaction output value. If the bitcoin miners' have a deterministic mempool which prioritizes transactions respecting a "weak" local time order of transactions, it could become extremely unprofitable and unlikely for an attack to succeed. Any transaction spam time attack by broadcasting the incorrect Commitment Transaction is extremely high risk for the attacker, as it requires an immense amount of bitcoin and all funds commited in those transactions will be lost when the attacker fails.

9. Use Casaes

In addition to helping bitcoin scale, there are many uses for these types of transactions

- Instant Transactions. Bitcoin transactions are now nearly instant with any party on this network. It is possible to pay for a cup of coffee with direct non-revocable payment in milliseconds to seconds.
- Exchange Arbitrage. There is presently incentive to hold funds on exchanges to be ready for large market moves due to 3-6 block confirmation times. It is possible for

the exchange to participate in this network and for clients to move their funds on and off the exchange for orders nearly instantly. If the exchange does not have deep market depth and commits to only permitting limit orders close to the top of the order book, then the risk of coin theft becomes much lower. The exchange, in effect, would no longer have any need for a cold storage wallet. This may substantially reduce thefts and the need for trusted 3rd party custodians.

- Micropayments. Bitcoin blockchain fees are far too high to accept micropayments. With this system, micropayments using Bitcoin without a 3rd party custodian is now possible. It is possible to pay per-megabyte for internet service or per-article to read a newspaper.

10. Conclusions

Creating a network of micropayment channels enables bitcoin scalability, micropayments down to the satoshi, and near-instant transactions. These channels are real Bitcoin transactions, using the Bitcoin scripting opcodes to enable transfer of funds without outright counterparty theft, if some block soft-cap is in place and enforced as part of the miners' consensus rules.

If all Bitcoin transactions were on the blockchain, to enable 7 billion people to make two transactions per day, it would require 24GB blocks every ten minutes at best (presuming 250 bytes per transaction and 144 blocks per day). Conducting all global payment transactions on the blockchain today implies miners will need to do an incredible amount of computation, severely limiting bitcoin scalability and full nodes to a few centralized processors.

If all Bitcoin transactions were conducted inside a network of micropayment channels, to enable 7 billion people to make two channels per year with unlimited transactions inside the channel, it would require 133 MB blocks (presuming 500 bytes per transaction and 52560 blocks per year). Current generation desktop computers will be able to run a full node with old blocks pruned out on 2TB of storage.

With a network of micropayment channels whose payments are encumbered by timelocks and hashlock outputs, Bitcoin can scale to billions of users without custodial risk or blockchain centralization when transactions are conducted securely off-chain using bitcoin scripting, with enforcement of non-cooperation by broadcasting signed multisignature transactions on the blockchain.

Acknowledgements

Micropayment channels have been developed by many parties, and have been discussed on bitcointalk, the bitcoin mailing list, and IRC. The amount of contributors to this idea are immense and much thought have been put into this ability. Effort has been placed into citing and finding similar ideas, however it is absolutely not near complete. In particular, there are many similarities to a proposal by Alex Akselrod by using hashlocking as a method of encumbering a hub-and-spoke payment channel.

[TODO NOTE: still working on references]

Appendix A: Trusted Hub-and-Spoke Micropayment Networks

It is possible to create a hub-and-spoke micropayment network which has weak dependence upon a 3rd party, with much lower expiry times.

To spend from this alternative micropayment channel in a hub-and spoke network, there are multiple outputs from the Commitment Transaction. The first two include balances for Alice and Bob, the third output are the sends in progress. These sends are a 2-of-3 multi-signature output. The 3rd signature is the clearing house service.

If Alice wishes to send funds to Carol through Bob, using clearing house service Hector, Carol first gives Alice a signed message requesting funds using hash(R), derived from a 32-byte random data R. Then Alice creates a transaction whereby the output for the Commitment Transaction includes a 2-of-3 multisig between Alice, Bob and Hector. Alice also sends a separately signed message asserting the hash(R), the desired timeout, as well as the desired routing endpoint. This message is out-of-band from the bitcoin system. It should only be read by Bob and Hector to verify the end recipient and hash(R) and can be in whatever agreed upon format.

Bob then does the same with Carol by providing the same hash(R) and a 2-of-3 multisig to Hector. Carol then discloses R to Bob, who then discloses it to Alice. At this point, all parties double-spend their previous Commitment Transaction without the 2-of-3 multisig encumbered output if the transaction occurs within some predefined amount of time (e.g. 30 seconds). If any part of this transaction does not complete before the alloted time, Hector is notified and the transaction path is invalidated. Note that the desired timeout by each participant must be lower for the next hop.

The clearing house service is not contacted unless the transaction expired before it could finish.

It is possible to create multi-hop networks with a chain of 4 or 5 participants. It is also possible to use different clearing house services along this path, so long as one of the intermediary participants is willing to have that exposure. Since the clearing house service may be anyone, and may change along the path, the lock-in due to network effects are marginal.

Low latency clearing is possible when having a trusted 3rd party service. Since this 3rd party clearing house service is capable of colluding with one's counterparty to steal funds (as it is in a 2-of-3 multisig), only extremely small micropayments should be sent. This is not a problem because sends should be very close to free and splitting up a large transaction to hundreds or even thousands of tiny transactions should be inconsequential. If the clearing house service and counterparty can only collude to steal 0.000001 BTC at a time.

Under this model, the trusted clearing house service does not take full custody of funds. All funds are locked with a 2-of-3 multisig, so theft requires collusion of multiple parties.

Appendix B: Fixing Malleability

In order to create these contracts in Bitcoin without a 3rd party trusted service, Bitcoin must fix the transaction malleability problem. If transactions can be mutated, then signatures can be invalidated (as they are spending from a transaction ID which does not exist), thereby making refund transactions and commitment bonds invalidated. This creates an opportunity for hostile actors to use it as an opportunity for a negotiating tactic to steal coins, in effect, a hostage scenario.

Fixing malleability necessitates to make a soft-fork change to bitcoin. Older clients would still work, but miners would need to update. Bitcoin has had several soft forks in the past, including pay-to-script-hash (P2SH).

To mitigate malleability, it requires changing which contents are signed by the participants. This is achieved by creating new Sighash types. In order to accommodate this new behavior, a new P2SH type or new OP_CHECKSIG is necessary to make it a soft-fork rather than a hard-fork.

Either may be used, if a new P2SH was defined, it would use a different output script such as:

```
OP_DUP OP_HASH160 <20-byte hash> OP_EQUALVERIFY
```

Since this will always resolve to true provided a valid redeemScript, all existing clients will return true.

This opens the scripting system to construct new rules, including new signature validation rules. Two new sighashes would need to be constructed. The new sighash, SIGHASH_NORMALIZED would sign the normalized transaction ID as an input. A second sighash type, SIGHASH_NOTXID, will not sign any input transaction IDs. A third sighash type, SIGHASH_NOINPUT, will not sign any input transactions IDs nor sign the index.

SIGHASH_NORMALIZED may be used for any transaction which doesn’t require any unusual scripting and permits transactions to resolve malleability without significant change in behavior for current wallets interested in blockchain payments only. SIGHASH_NOTXID and SIGHASH_NOINPUT implies significant risk with address reuse and should only be used for transactions where it is necessary, such as in combination with an unbroadcasted SIGHASH_SINGLE or SIGHASH_ANYONECANPAY, or where an unbroadcasted transaction needs to be re-used. The transaction itself still uses non-normalized Transaction IDs, but signs with the normalized Transaction ID if it is SIGHASH_NORMALIZED. In the event a transaction is mutated, a transaction must be updated to use the new Transaction ID, but does not need to be re-signed.

This allows one to be assured that one’s counterparty cannot invalidate entire trees of chained transactions of potential contract states which were previously agreed upon, using transaction ID mutation. With the new sighash flags, it is possible to spend from a parent transaction even though the transaction ID has changed.

Further, and just as importantly, it permits participants to sign spends of transactions without knowing the signatures of the transaction being spent. By solving malleability in the above manner, two parties may build contracts and spending on transactions without either party having the ability to broadcast that original transaction on the blockchain until both parties agree. With the new sighash types, participants may build potential contract states and potential payout conditions and agree upon all terms, before the contract may be paid, broadcast, and executed upon without the need for a trusted 3rd party.

References

[1] Satoshi Nakamoto. *Bitcoin: A Peer-to-peer Electronic Cash System.* https://bitcoin.org/bitcoin.pdf Oct 2008.

[2] https://en.bitcoin.it/wiki/Contracts#Example_7:_Rapidly-adjusted_.28micro.29payments_to_a_pre-determined_party

[3] https://bitcoinj.github.io/working-with-micropayments

[4] Leslie Lamport. *Time, Clocks, and the Ordering of Events in a Distributed System*. Communications of the ACM 21, 7 (July 1978), 558-565.

[5] Alex Akselrod. *Draft*. https://en.bitcoin.it/wiki/User:Aakselrod/Draft Mar 2013.

[6] Alex Akselrod. ESCHATON. https://gist.github.com/aakselrod/9964667 April 2014.

[7] Peter Todd. *Near-zero fee transactions with hub-and-spoke micropayments.* http://sourceforge.net/p/bitcoin/mailman/message/33144746/ Dec 2014.

[8] BitPay. *Impulse.* http://impulse.is/impulse.pdf Jan 2015.

[9] Peter Wuille. *BIP 0032: Hierarchical Deterministic Wallet*. https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki Feb 2012.

[10] Ilja Gerhardt and Timo Hanke. *Homomorphic Payment Addresses and the Pay-to-Contract Protocol.* http://arxiv.org/abs/1212.3257 Dec 2012.

[11] Nick Szabo. *Formalizing and Securing Relationships on Public Networks.* http://szabo.best.vwh.net/formalize.html Sept 1997

[12] Peter Todd. *CHECKLOCKTIMEVERIFY.* http://blog.viacoin.org/2014/12/25/petertodd-dev-update.html Dec 2014