A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels

Christian Decker   Roger Wattenhofer

Distributed Computing Group, ETH Zurich
cdecker@ethz.ch   wattenhofer@ethz.ch

Abstract. Bitcoin does not scale, because its synchronization mechanism, the blockchain, limits the maximum rate of transactions the network can process. However, using off-blockchain transactions it is possible to create long-lived channels over which an arbitrary number of transfers can be processed locally between two users, without any burden to the Bitcoin network. These channels may form a network of payment service providers (PSPs). Payments can be routed between any two users in real time, without any confirmation delay. In this work we present a protocol for duplex micropayment channels, which guarantees end-to-end security and allow instant transfers, laying the foundation of the PSP network.

1 Introduction

Credit card companies process a growing number of transactions, currently more than 10,000 per second. In contrast, Bitcoin currently handles about one transaction per second. Bitcoin's turnover is growing, and ultimately Bitcoin may become a viable payment alternative. However, can Bitcoin scale to match the throughput of credit cards, or even an envisioned world of millions of micropayments per second?

The answer to this question is astonishingly negative. In order to verify whether a new transaction is valid, and in order to bootstrap new peers, every peer in the Bitcoin network stores every transaction ever. The size of an average transaction is 500 bytes, so with 1 transaction per second, every Bitcoin peer now needs almost 20 GB of additional storage each year. A turnover of 500 transactions per second would require 10 TB of additional disk space per year, which is at the limit for a consumer.

A bigger problem is processing power. Checking the signatures of each transaction (mostly because of disk seek time) takes about 5 ms, so with current machines we cannot hope to scale beyond 200 transactions per second.

Every node in the bitcoin network is informed about every transaction, multiple times because of the fault-tolerant gossip process. Assuming a common end-user bandwidth of 10 Mbit/s, then the rate peers can receive transactions is limited to approximately 1,000 transactions per second. Finally, while peers may individually be able to receive and process up to 200 transactions per second, the synchronization mechanism underlying Bitcoin is susceptible to latency, and does not work with transaction rates above 100 transactions per second [6].

In summary, Bitcoin in its current form will have a hard time scaling beyond 100 transactions per second, because of storage, processing, latency, and bandwidth. The problem of Bitcoin is its reliance on a synchronized global state, the replicated *blockchain*.

In this paper, we propose to reduce the reliance on the blockchain to further decentralize the architecture of Bitcoin. We believe that the blockchain should only be used to establish long lived point-to-point channels between parties over which an arbitrary number of transfers can be performed. These transfers are no longer Bitcoin transactions that are committed to the blockchain, instead they rely on off-blockchain transactions that summarize any number of transfers between two parties. The blockchain is only involved during the setup and the closure of such a channel, while the vast majority of updates is never committed to the blockchain.

Towards this goal we present a duplex micropayment channel protocol. Duplex micropayment channels are established between *payment service providers* (PSPs). PSPs are the equivalent autonomous systems in the Internet, routing transfers between end users, possibly over multiple hops, guaranteeing end-to-end security and enabling real-time transfers. Unlike Bitcoin transactions, which take minutes to be confirmed, transfers over our duplex micropayment channels are final and can be accepted without further confirmations, enabling real-time payments, and a truly scalable future Bitcoin.

2 Bitcoin

In this section we give a short overview on the basic Bitcoin protocol. Specifics necessary for the duplex micropayment channel are discussed in detail later on. Bitcoin is a distributed system running on a homogeneous peer-to-peer network. Peers in the network collectively maintain a global state, known as the ledger, which tracks bitcoins and their associations. The fundamental data unit tracked by the network is the *output*, a tuple consisting of a value denominated in bitcoins and an output script. The output script sets up a claiming condition that has to be satisfied in order to claim the bitcoins associated with the output. The most common case is that a signature matching an address is required. Hence, the balance of an address is the sum of all outputs whose output scripts require that address' signature.

The only operation that may modify the global state is a *transaction*. A transaction claims one or more previously unclaimed outputs and creates new outputs. By providing inputs matching the output script, the creator of the transaction proves that she is allowed to claim the output. A transaction may redistribute the sum of values to new outputs and may set up arbitrary claiming conditions for the outputs.

In order to apply a transaction to the replicas of the ledger, the transaction is flooded in the network. When a node in the network receives a transaction the node first verifies the signatures of the transaction and, if valid, the transaction is applied to the local replica. For each input the script is executed with the input from the claiming transaction. If all scripts return true, the outputs were not claimed by a previous transaction, and the sum of new output values is smaller than the sum of claimed output values the transaction is valid. Due to the distributed nature of the system, the order in which transactions are applied is not identical across peers, and peers may disagree about the validity of a transaction, e.g., if two or more transactions attempt to claim the same output, the validity depends on the order they are seen by the peers.

Bitcoin eventually resolves inconsistencies by electing one peer as leader, which may then impose its changes to other peers, by sending a *block* containing all transactions it accepted since the last block. Each block contains a reference to its predecessor, incrementally building the *blockchain*, a shared history of all transactions that were applied. Transactions that are included in a block of the blockchain are said to be committed or confirmed. Leader election happens only rarely at random intervals; on expectation conflicts are resolved every 10 minutes. This is on purpose in order to minimize collisions in which multiple contradicting blocks are broadcast. However, it also introduces a long delay until a transaction is confirmed.

3 Building Blocks

In the following the concepts and sub-protocols used in this work are described in more detail.

3.1 Bitcoin Contracts

Off-blockchain transaction protocols are an example of *cryptocurrency contracts*. Contracts allow business logic to be encoded in Bitcoin transactions which mutually guarantee that an agreed upon action is performed. The blockchain acts as conflict mediator should a party fail to honor an agreement.

In this work we concentrate on off-blockchain transaction protocols. Furthermore we limit the description to two parties, A and B, i.e., the two ends of the duplex micropayment channel. We denote the effective balances in the protocols or sub-protocols as σ_A and σ_B. Since the balances may change we denote the balances after update i as $\sigma_{A,i}$ and $\sigma_{B,i}$.

The main concern with off-blockchain transactions is to ensure that no party may renege on the agreement, possibly stealing funds from the other party. While on-blockchain transactions ascertain that a transaction has been committed before starting the next trade, a contract may last a long time and all parties have to ensure that they cannot be defrauded. A protocol is required in order to achieve mutual assurance that the latest update to the agreement is the one that will eventually be committed, and thus to invalidate any previous agreements. That is, each update creates a new set of transactions that supersede the previous update. At any time only one set of transactions may be released to Bitcoin and will be confirmed.

The protocol has to be carefully designed to avoid any possibility for fraud. Fraudulent behavior of a party may result in funds being stolen and funds being inaccessible either temporarily or permanently. Our protocol guarantees that funds are eventually refunded.

We assume that a suitable solution for transaction malleability [7] has been implemented [1,15]. Since transactions refer to the outputs they spend by the hash of the transaction which created the output, any change causing the hash to change will unlink the transactions. The protocols in this work use chains of transactions with multiple signatures. Since ECDSA signatures are inherently malleable, anyone with the ability to re-sign a transaction may invalidate subsequent transactions. If deterministic and non-malleable signature schemes are used instead, all of our presented schemes can still be implemented securely, although they will become more complex. Most of the solutions aim to normalize transaction hashes by removing the signatures before hashing. This also enables the creation of transactions that spend outputs created by a transaction that is partially signed.

3.2 Timelocks and Invalidation

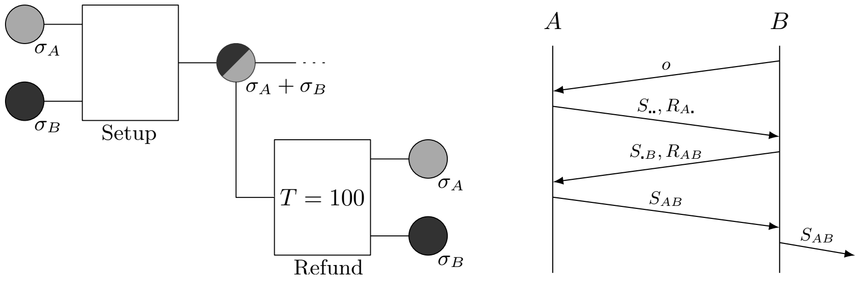
Bitcoin provides a mechanism to makes transactions invalid until some time in the future: *timelocks*. In addition to the validity conditions mentioned in the Section 2, a transaction may specify a locktime: the earliest time, expressed in either a Unix timestamp or a blockchain height, at which it may be included in a block and therefore be confirmed.

Peers in the network discard transactions with future timelocks. Any block including the transaction, that appears at a lower height or before the specified time, is deemed invalid. Timelocks can be used to replace or supersede transactions: a transaction with timelock T can be superseded by another transaction, spending some of the same outputs, with timelock $T' < T$ and ensuring that the superseding transaction is broadcast to the network before the superseded transaction becomes valid.

Timelocks are transitive, i.e., a transaction spending an output created by a timelocked transaction will only be valid once the timelocked transaction is committed. Hence a transaction spending timelocked outputs has an effective timelock matching the maximum timelock of any transaction it depends on.

In order to update the contract, e.g., to increase the value one party will receive in the end, it is necessary to invalidate or replace transactions during the execution, ensuring that only the latest update is valid. Throughout the protocol two invalidation techniques are used:

- *Replace by timelock*: both parties hold fully signed transactions, with different bitcoin allocations, of which only one may be committed. All transactions have a timelock in the future. Only the transaction with the smallest timelock will eventually be committed, i.e., it is released before any other transaction becomes valid.



(a) Final setup structure.

(b) Protocol sequence diagram.

Fig. 1: Setup creating a multisig output of value $\sigma_A + \sigma_B$ from two outputs of value σ_A and σ_B. The refund transaction is timelocked and only valid after T=100. The sequence of transaction exchanges detailed on the right ensures the security of the setup. Subscripts represent the signatures by A and B or a **.** if a signature is missing.

– *Replace by incentive*: one party has multiple fully signed transactions, with different values transferred to it, of which only one may be committed. The party will commit the transaction transferring the highest amount to it.

In order to guarantee that replace by timelock is secure the difference between timelocks that supersede each other has to be at least ΔT. Due to the confirmation rate of Bitcoin we chose ΔT to be 1 hour. To simplify the notation we express timelocks as multiples of ΔT and use offsets such that the protocol starts at $T = 0$.

3.3 Shared Accounts

When an output can be claimed by providing a single signature it is called a *singlesig output*. In contrast the script of *multisig outputs* specifies a set of n public keys and requires m-of-n (with $m \leq n$) valid signatures from distinct matching public keys from that set in order to be valid.

In the 2-of-2 case two parties, A and B, have to sign transactions spending the output. This is akin to a shared account where any transaction spending the common funds must be signed off by both parties. If both A and B have supplied σ_A respectively σ_B bitcoins to a multisig output, the output's value is $\sigma_A + \sigma_B$. Of this total value we say that A effectively owns σ_A and B effectively owns σ_B, despite both signatures being required to spend the output.

Once a multisig output has been created and committed to the blockchain, A and B are guaranteed that the funds of the output may not be spent by either of the parties without both agreeing. As such the creation of a multisignature output is often used in order to setup a contract.

In order to securely create a shared account (multisig output) two transactions are needed: a *setup transaction* and a *refund transaction*. The setup transaction claims some funds from singlesig outputs owned by A and B, and creates the multisig output. The refund transaction ensures that the funds are eventually refunded should one party disappear and not provide the necessary signatures to spend the multisig output.

Figure 1 shows the setup of a shared account coordinated by A. First B sends a list o of outputs it desires to add to the shared account, for a total value of σ_B bitcoins. A creates an unsigned setup transaction that claims both o and its own outputs, with a value of σ_A bitcoins, and creates a 2-of-2 multisig output requiring signatures from both A and B to be spent. In addition it creates a refund transaction that spends the newly created multisig output and transfers σ_A to a singlesig output requiring A's signature and σ_B to a singlesig output requiring B's signature. The refund transaction has a timelock some time in the future, making it invalid until that time.

The protocol sequence diagram in Figure 1 shows the order in which messages are exchanged. A adds its signature to the refund transaction and sends both the refund transaction and the unsigned setup transaction to B. Upon receiving the transactions, B verifies that the refund transaction eventually returns its funds and adds its signature to both transactions. B now has a valid refund transaction and a partially signed setup transaction. Both transactions are returned to A which adds the missing signature to the setup transaction, making all transactions fully signed. The setup transaction is then released to the Bitcoin network and committed to the blockchain. This locks the funds until the refund returns them to the respective owners or until both parties agree on a different division of the funds, signing another transaction that supersedes the refund.

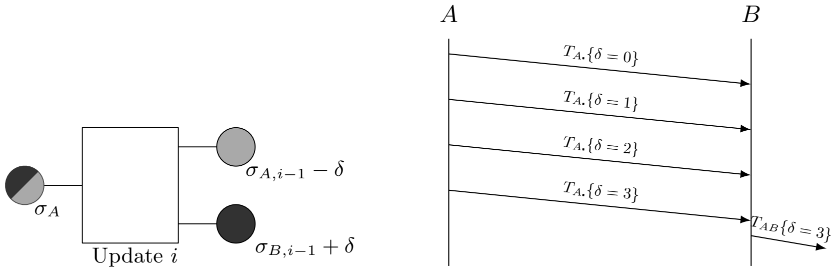
3.4 Simple Micropayment Channels

Simple micropayment channels, first introduced by Hearn and Spilman [9], are contracts that can be established between two parties, a sender and a receiver. Once a micropayment channel is established, the sender can send incremental micropayments to the receiver. The channel has a limit determined by the sender upon the channel's creation. Once the limit is consumed, i.e., transferred entirely to the receiver, the channel is closed.

The micropayment channel can be created by setting up a shared account, as described in the previous section, between the sender and the receiver. The sender A funds the channel with σ_A, whereas the receiver does not contribute, i.e., σ_B is 0. We denote $\sigma_{A,i}$ and $\sigma_{B,i}$ to be the owned amounts after the i^{th} update by A and B respectively.

In order to perform an incremental micropayment of value δ at time $i+1$, A creates a *micropayment update transaction* spending the multisig output and transferring $\sigma_{A,i+1} = \sigma_{A,i} - \delta$ and $\sigma_{B,i+1} = \sigma_{B,i} + \delta$ to A and B respectively.

The update transaction is signed by A and sent to the receiver B. At this point the receiver could add its own signature and broadcast it to the Bitcoin network, committing it to the blockchain. However, normally the transaction is not broadcast. Instead the receiver accepts new update transactions, which transfer a larger amount to it. Only one of the update transactions may be committed to the blockchain since they all spend the same output. The receiver



(a) Micropayment channel structure. (b) Payment channel sequence.

Fig. 2: The structure of the payment channel consists of a single transaction splitting the value of a multisig output among the participants. In this case A funded the channel and may send to B and δ is the sum of increments.

is incentivized to only use the latest update as it is the one paying out the maximum amount.

Eventually (i) all the initial funds $\sigma_{A,0}$ are transferred to B, (ii) both parties agree on closing the channel, or (iii) the refund time from the setup is approaching, triggering B to close the channel. To close the channel, B broadcasts the last update transaction which supersedes the refund transaction.

Note that such a micropayment channel is intrinsically unidirectional, i.e., the amount that the receiver is assigned in update transactions must be strictly increasing, otherwise the receiver might release an earlier update, which pays out a higher amount.

3.5 Atomic Multiparty Opt-In

In the shared account setup protocol, great care had to be taken about the order in which signatures were added, to avoid situations where funds could be locked in indefinitely. *Atomic multiparty opt-in* is an off-blockchain protocol that enables multiple parties to negotiate the creation of a complex structure of transactions, built on top of existing multisig outputs, without having to worry about the order in which the signatures are added. The structure can be negotiated openly since parties activate, or opt in, only after it is secure.

The atomic multiparty opt-in protocol uses an *opt-in transaction* O which claims a multisig output and creates a new multisig output, called the *root output*. Subsequent transactions spend the root output and thus are valid only if the opt-in transaction is valid, i.e., when all parties sign the opt-in transaction. This also obviates any refund addresses attached to intermediate outputs, which would be needed if each subsequent transaction were negotiated independently.

One party creates an unsigned opt-in transaction which spends a multisig output, requiring signatures from all participants, and creates one or more root outputs. The participants then collaborate to create the updated version of the contract, openly sharing any necessary transactions and signatures. As soon as all parties are content with the contract they sign the opt-in transaction,

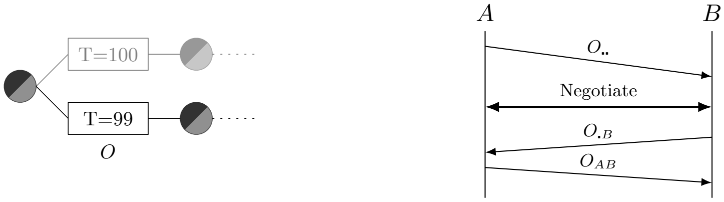


Fig. 3: Opt-in structure to update an existing contract. The version on top is superseded by the lower version. Transactions attached to the root outputs on the right are negotiated openly, with the opt-in transaction determining validity.

making it valid. The fully signed opt-in transaction is then exchanged among all participants to ensure that all parties can enforce the decision.

The atomic multiparty opt-in can be used in two ways: (i) to initially set up a contract starting from a multisig output owned by the participants, or (ii) to update an existing contract by building a structure that spends the root output of an outdated contract. In the latter case, depicted in Figure 3, it is necessary to enforce that only the new version is valid by using a smaller timelock.

The protocol is off-blockchain as its transactions are only committed to the blockchain if one party defects. Notice that the party signing last may unilaterally decide whether to sign and commit or not. It is therefore advisable to use the multiparty opt-in exclusively in idempotent updates, i.e., when the value that is paid out to the parties does not change depending on whether or not the opt-in is committed.

3.6 Hashed Timelock Contracts (HTLC)

Hashed Timelock Contracts, or HTLCs, are contracts that require the recipient of a payment to reveal a secret in order to claim an output before it is refunded to the sender. The ability of the recipient to claim the output is therefore conditioned on its ability to reveal the secret.

This can be used to enable end-to-end security in a multi-hop scenario, in which a single payment is forwarded through multiple parties. In this scenario, B requests a payment from A and specifies the hash $h(S)$ of a secret S, which will be used to unlock the payment. A creates an HTLC output from a shared account with the next hop on the path to B. The HTLC output sets up the claiming condition as shown in Figure 4: either the next hop provides S' s.t. $h(S) = h(S')$ and a valid signature from both parties, or both parties must sign the transaction spending the HTLC output. This procedure is repeated by each node on the path until B is reached. B then releases S to its previous node, claiming the HTLC output, and giving the previous node the ability to claim the previous HTLC output. This is repeated until the secret is revealed to A, thus completing the transfer.

For each hop there is a sender H_A and a receiver H_B and they share a multisig output that is used for the transfer. The HTLC output is created by an

```
OP_IF
   2 <A pubkey> <B pubkey> 2
   OP_CHECKMULTISIG
OP_ELSE
   OP_HASH160 <S hash>
   OP_EQUALVERIFY
   2 <A′ pubkey> <B pubkey> 2
   OP_CHECKMULTISIG
OP_ENDIF
```

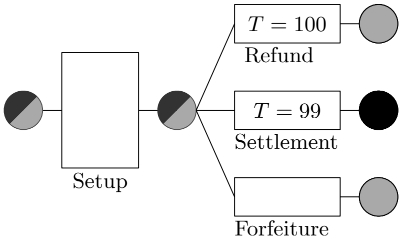


Fig. 4: HTLC output script and structure. The first branch is a normal multisig script while the second branch requires a secret and both signatures.

HTLC setup transaction, claiming the multisig output. During the execution of the protocol up to three transactions are created that may claim the HTLC output: a *refund transaction*, a *settlement transaction*, and a *forfeiture transaction*. The refund transaction is identical to the one from the shared account setup and ensures that H_A is refunded should H_B not cooperate. The settlement transaction performs the transfer from H_A to H_B if the latter reveals the secret. Finally, the forfeiture transaction is used to guarantee that H_A is refunded even if the secret is eventually revealed. The last scenario is used to remove the HTLC output before the refund becomes valid, i.e., when both parties agree to free the funds locked in the HTLC output without performing the transfer.

The sender creates the HTLC setup transaction and all three transactions spending the HTLC output and signs refund transaction, forfeiture transaction and settlement transaction. The settlement transaction uses the *else*-branch of the script, which uses a separate *HTLC signing key* for the sender. This is necessary since otherwise H_B could simply use the same signature in the *if*-branch, since signatures are valid for both branches. The partially signed refund, forfeiture and settlement transactions are then sent to the receiver which adds its signature to the refund and sends it back. The sender signs the HTLC setup transaction and sends it to the receiver, which may attempt to claim the HTLC output unilaterally by providing its signature and the secret to the settlement transaction.

The lifetime of the HTLC output is limited by the refund transaction's timelock, and should H_B want to claim it, it must release the settlement transaction before the refund is valid. While this protocol works when committing transactions directly to the blockchain, its main use is in off-blockchain transactions.

In order to be usable in off-blockchain transactions, the timelock of the refund must be later than those in refund transactions attached to the root outputs, i.e., it must be guaranteed that H_B indeed has time to claim the HTLC output on the blockchain before the refund transaction becomes valid. Should the receiver disclose the secret S to the sender, then both parties can agree on removing the HTLC output and instead add its value to another output that directly transfers to the receiver. On the other hand, should H_B not be able to disclose S then it may decide to forfeit the HTLC output. In this case both parties sign the forfeiture transaction with no timelock, spending the HTLC output back to the sender. Once the sender has a fully signed forfeiture transaction, the receiver may not claim the HTLC output anymore since the forfeiture transaction is valid before the settlement transaction.

The HTLC output can be attached to an existing micropayment channel, the sender would simply send a micropayment update transaction which includes the HTLC output of value δ.

4 Duplex Micropayment Channel

The secure setup, the micropayment channel and the hashed timelock contract alone enable the use multi-hop micropayments with end-to-end security. However setting up two independent micropayment channels between two peers, one for each direction between, is fairly limited. Each channel is unidirectional and is limited by the amount of bitcoins locked in during the setup by the sender. Once the limit has been consumed, the channel has to be torn down and a new one created, incurring time delay and cost of committing several transactions to the blockchain.

While this cannot be avoided on connections at the edge of the network in which a majority of payments flows in one direction, connections in which payments flow in both directions may take advantage from resetting their channels once the limit is consumed. For example, consider the channels C_{AB} from A to B and C_{BA} in the opposite direction, each initially funded with 1 coin. The limit of C_{AB} may have been consumed, and C_{BA} has a residual of 0.5 bitcoins. No further transfer from A to B can be performed despite A having a non-zero balance on the C_{BA} channel, i.e., when considering both channels the balances are $\sigma_A = 0.5$ and $\sigma_B = 1.5$. In order to enable future transfers from A to B both parties could agree to reset the channel, i.e., new channels C'_{AB} and C'_{BA} are created and funded with 0.5 and 1.5 bitcoins respectively. Notice that in both the depleted case and the reset case A and B own the same amount of bitcoins, but the channel their share is bound to has changed.

In the following we describe the duplex micropayment channel protocol that enables atomically resetting a set of channels. By doing so we enable the initial funds to be transferred over the duplex channel an arbitrary number of times, and hence reduce the necessity to commit to the blockchain.

A duplex micropayment channel (DMC) is established between two parties A and B. The protocol establishes pairs of simple micropayment channels, one for each direction between the two parties. In order to reset the channels the protocol generates a sequence of pairs of unidirectional micropayment channels. We use $C_{AB,j}$ and $C_{BA,j}$ to indicate the simple micropayment channels in the j^{th} pair of channels. Furthermore we define $\sigma_{X,j,i}$ to be the amount that the pair of micropayment channels would transfer to party $X \in \{A, B\}$ if they were committed to the blockchain after update i in the pair j.

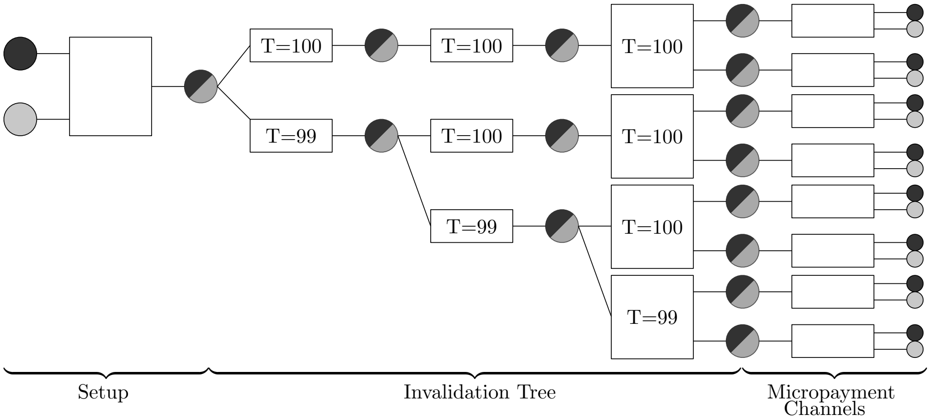


Fig. 5: A full example of the duplex micropayment channel with $n = 1$ and $d = 3$, allowing up to 4 resets.

4.1 Structure

The fundamental structure of the DMC is the *invalidation tree*. The invalidation tree is a tree in which multisig outputs are the nodes of the tree, connected by transactions as edges. Each transaction in the tree is given a timelock, such that there is a unique minimal timelock among all sibling transactions, i.e., transactions sharing the same parent output. By the replace by timelock rule, only one path from the root of the tree is therefore first valid, i.e., the path with the minimal timelocks for each level in the tree. Hence as long as all timelocks are in the future, we can invalidate an entire subtree, by adding a new transaction spending that subtree's root output, with a smaller timelock than all existing transactions. We define two times T_{max} and T_{min} in terms of locktime. All refund transactions are set to have locktime T_{max}, forcing parties to commit the protocol's state to the blockchain before that time in order to avoid triggering the refunds. T_{min} is the minimum timelock that is going to be used in the invalidation tree to replace other transactions. The time from the channel creation to T_{min} is referred to as the channel's lifetime.

The number of replacement by timelock is limited by $n = (T_{max} - T_{min})/\Delta T$. Therefore each multisig output in the invalidation tree may have at most n outgoing transactions which replace each other. Furthermore, due to the transitivity of timelocks, the full range may not be available as adding a timelock that is lower than one of its parent transactions has no effect: all transactions with a lower timelock become valid simultaneously, resulting in a race condition. For simplicity we limit the depth of the tree to d. This limits the number of transactions that have to be committed to the blockchain should one party defect.

The depth d, the number of replacements in the tree n and time until funds are refunded T_{max} are parameters to the duplex micropayment channel and are negotiated before the channel is created. T_{min} can be derived from T_{max}, n and ΔT, which is a system parameter.

Furthermore, knowing n and d allows the enumeration of all branches in the tree. A branch can be represented as a string of length d, the alphabet $\{T_{min}, ..., T_{max}\}$ and the elements are increasing. Thus every branch has a unique successor that directly invalidates it. This facilitates the negotiation of which branch to select next.

The internal nodes of the invalidation tree are individual multisig outputs, while the leafs of the tree are pairs of multisig outputs. On the leaf outputs a pair of simple micropayment channels is built, one transferring from A to B and the other one in the opposite direction.

Multi-hop payment flows result in HTLC outputs being attached to the simple micropayment channel matching the direction of the flow. The timelock of the transactions spending the HTLC outputs are larger than T_{max}. This ensures that the micropayment channel creating the HTLC have been committed to the blockchain and replace by timelock can be performed. The period between T_{max} and the last HTLC output being claimed is referred to as *conflict resolution phase*.

4.2 Setup

The setup initiates the micropayment channel between two parties by locking in the initial funds into a shared account. The shared account creation subprotocol from Section 3.3 is used to create the multisig output. Both parties exchange a set of singlesig outputs they would like to contribute to the channel and create the setup transaction. The initial funds from A and B are denoted as $\sigma_{A,0,0}$ and $\sigma_{B,0,0}$ since there were no resets and no updates yet. The refund transaction has a timelock of T_{max}. It transfers the funds back to their owners if no other agreement is committed first. Since the setup transaction is committed in the blockchain it is safe to build upon the multisig output. Committing the transaction may take several minutes and the channel is not operational until it is committed.

4.3 Reset

The reset process takes care of building a new branch of the invalidation tree and setting up the micropayment channels. This includes the first branch starting from the shared account the setup created. A reset is triggered after the initial setup, as well as when the limit of one of the simple micropayment channels is depleted. Assuming that the limit of A's channel $C_{AB,j}$ is consumed and therefore requires a reset. A is said to coordinate the reset: it will no longer perform updates to its channel $C_{AB,j}$ and send a *reset request* to the B. Upon receiving the reset request, B stops performing updates to its channel $C_{BA,j}$ and sends a *reset response*. The reset response signals to A that B is willing to perform the reset and that no further updates to $C_{BA,j}$ will be performed and that the value transferred by the two simple micropayment channels $\sigma_{A,j,i}$ and $\sigma_{B,j,i}$ will not change.

Upon receiving the reset response, A can proceed to build the next branch ending in two multisig outputs. The values of the two multisig outputs are

$\sigma_{A,j+1,0} = \sigma_{A,j,i}$ and $\sigma_{B,j+1,0} = \sigma_{B,j,i}$, i.e., each multisig output is virtually owned by one party and its value represents the share the owner would get if the current branch were to be committed. On top of the leaf multisig outputs two new simple micropayment channels $C_{AB,j+1}$ and $C_{BA,j+1}$ are built with respective refund transactions. The branch is negotiated as an instance of the atomic multiparty opt-in protocol, with the transaction spending the existing output from the previous branch as opt-in transaction and the remainder of the branch as subsequent structure. A may sign the entire branch where necessary, except the opt-in transaction, which may only be signed once B has signed the refund transactions for the simple micropayment channels, therefore assuring that funds will not be locked in indefinitely.

The atomic multiparty opt-in ensures that either both agree on switching to the new branch or the old branch remains active. In both cases the same amounts are transferred to the two parties and updates to the micropayment channels $C_{AB,j+1}$ and $C_{BA,j+1}$ resume only once both parties have a fully signed opt-in transaction.

4.4 Teardown and Commit

Eventually the duplex micropayment channel needs to be closed and the summary of the channel committed to the blockchain. The closure of the duplex micropayment channel can be triggered by agreement or by the end to the channel's lifetime. Either both parties agree on the summary, or they disagree and do not collaborate. In the first case they may simply create a *teardown transaction*, which transfers $\sigma_{A,j,i}$ to A and $\sigma_{B,j,i}$ to B, assuming update i is the latest update in the current round j. The teardown transaction is not timelocked and directly spends the multisig output created in the setup process, hence it can be committed to the blockchain immediately. The process simply involves one party creating the teardown transaction, both parties signing it and committing it to the blockchain. HTLC outputs which have not been removed by agreement can be copied over to the summary transaction such that the same timelocks and resolution rules apply.

In the case parties do not agree on the summary of the channel, they still have the latest branch of the invalidation tree that guarantees eventual conflict resolution. Before the refunds become valid the branch is submitted to the Bitcoin network and will be committed to the blockchain. Unlike the commit using a summary transaction, which requires just a single transaction to be committed, the resolution by tree branch requires up to $d + 2$ transactions, hence we limit on the depth of the tree.

4.5 Refresh

In the case two parties have an existing duplex micropayment channel its lifetime may be extended using the refresh process. Analogously to the reset subprotocol, both parties stop updating the existing duplex micropayment channel by exchanging *refresh request* and *refresh response* messages, thus flushing pending changes. The parties agree on new parameters T_{max} and T_{min} determining the new channel's lifetime. One party creates an opt-in transaction creating a new root output and a refund transaction with a timelock of T_{max} transferring $\sigma_{A,j,i}$ and $\sigma_{B,j,i}$ to their respective owners. Both parties then perform the atomic multiparty opt-in protocol using the opt-in transaction and the refund as subsequent structure. The opt-in transaction is then published on the Bitcoin network and committed to the blockchain, invalidating the entire invalidation tree built on the old root output.

Special care has to be taken with HTLC outputs as these may time out during the new channel's lifetime. The HTLC outputs are copied over to the opt-in transaction, and their resolution is handled on the blockchain.

The refreshed duplex micropayment channel is operational immediately, since the opt-in transaction is guaranteed to be eventually confirmed, i.e., no party may double-spend the old root output.

In addition funds can be removed and added during the refresh process. Funds can be removed adding singlesig outputs to the opt-in transaction that pay out part of a party's balance to one of its addresses, that party's share of the channel is then reduced accordingly. In order to add funds to the channel, a multisig output owned by both parties has to be created ahead of time using the protocol in Section 3.3 so that during the refresh the outputs are committed to the blockchain. This multisig output is then also claimed by the opt-in.

5 Routing Payments

A channel between two payment service providers (PSPs) can be established once; it has a lifetime of hundreds of days before it is either torn down or refreshed. The setup requires a single transaction that is committed to the blockchain locking in the initial funds, while the teardown requires a single transaction committed to the blockchain. In the case the two parties do not collaborate to close the channel, at most d transactions from the invalidation tree and two micropayment updates have to be committed to the blockchain. The amount of bitcoins transferred is only limited by the number of resets and the initial funds parties contribute to the channel. A channel with $n = 46$ and $d = 11$ results in $1.48 \cdot 10^{11}$ resets. If such a channel is initially funded with 1 bitcoin, the channel can be used to transfer a total of 148 billion bitcoins, an equivalent of 35.3 trillion USD at today's exchange rate, twice the US national debt. Notice that both n and d can be chosen arbitrarily, further extending the amount transferable by a channel.

By adding HTLC outputs to the micropayment channels, instead of sending the increment directly, the payment can be end-to-end secured so that the recipient of a payment has to confirm reception. The final recipient communicates the secret out of band to the sender of the payment. Each hop along the route from the sender to the recipient will create HTLC outputs transferring the funds only upon receiving the secret, which is only released once the final recipient is assured that the total is transferred.

6 Related Work

Bitcoin was introduced by Nakamoto in 2008 [11] and has since enjoyed a rapid growth both in value as in transaction volume. However, the design of Bitcoin intrinsically limits the rate it can process transactions. Barber et al. [4] identified problems with data retention, which later were adopted to create the simplified payment verification, using filtering nodes for mobile clients. An analysis of the information propagation [6] showed that the probability of blockchain forks rapidly increases with increasing transaction rates and the eventually the network is no longer able to resolve conflicts. Eyal et al. [8] further show how miners may use the propagation delay in the network as a force multiplier.

The GHOST protocol [14] allows an increase of the block generation rate by reusing blocks that are not in the main blockchain. Although mainly aimed at enabling innovation, Back et al. [2] propose dividing the single Bitcoin network into smaller networks that can operate independently. Discoin and PeerCensus [5] decouple the confirmation of transactions from the block generation and guarantee strong consistency. The slow confirmation also prevents a number of real-life uses of Bitcoin, as fast payment can be double-spent and not be detected for some time [3,10,13]. Our proposal enables secure end-to-end payments that do not require being confirmation in the blockchain, hence enabling true micropayment that clear in real-time.

Simple micropayment channels were introduced by Hearn and Spilman [9]. Finally the Lightning Network by Poon and Dryja [12], also creates a duplex micropayment channel. However it requires exchanging keying material for each update in the channels, which results in either massive storage or computational requirements in order to invalidate previous transactions. In our proposal the two channels operate independently allowing fully asynchronous operation between resets. Lightning renews the whole transaction structure on every update, requiring synchronous updates and high bandwidth consumption. Furthermore the Lightning protocol cannot be decomposed into smaller units that can be analyzed in isolation, making the security analysis difficult and resulting in complex implementations.

7 Conclusion

Duplex micropayment channels solve a multitude of problems. For one they enable near-infinite scalability for digital payments based on Bitcoin. Bitcoin transactions are no longer used directly to transfer bitcoins from a sender to a recipient, instead they are used to setup micropayment channels and handle conflict resolution. The actual transfers are now handled at a higher level through a network of payment service providers. The payments are end-to-end secure thanks to the use of hashed timelock contracts, ensuring transfers between hops are only performed if the intended recipient receives its payment. Unlike Bitcoin, which requires a long confirmation process, transfers on a network of duplex micropayment channels are secure from being reverted. Thus a payment network using duplex micropayment channels is a far better fit for real-time scenarios, e.g., buying a coffee, since transfers can be performed at the same speed messages are passed in the Internet. With a network of payment service providers, Bitcoin can support true micropayments with minimal fees at unprecedented scale, and where the transfers clear in real-time.

References

1. Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Łukasz Mazurek. How to deal with malleability of bitcoin transactions. *arXiv preprint arXiv:1312.3230*, 2013.
2. Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. Enabling blockchain innovations with pegged sidechains, 2014.
3. Tobias Bamert, Christian Decker, Lennart Elsen, Samuel Welten, and Roger Wattenhofer. Have a snack, pay with bitcoin. In *IEEE Internation Conference on Peer-to-Peer Computing (P2P), Trento, Italy*, 2013.
4. S. Barber, X. Boyen, E. Shi, and E. Uzun. Bitter to better—how to make bitcoin a better currency. *Financial Cryptography and Data Security*, 2012.
5. Christian Decker, Jochen Seidel, and Roger Wattenhofer. Bitcoin meets strong consistency. *arXiv preprint arXiv:1412.7935*, 2014.
6. Christian Decker and Roger Wattenhofer. Information propagation in the bitcoin network. In *IEEE International Conference on Peer-to-Peer Computing (P2P), Trento, Italy*, September 2013.
7. Christian Decker and Roger Wattenhofer. Bitcoin Transaction Malleability and MtGox. In *19th European Symposium on Research in Computer Security (ESORICS), Wroclaw, Poland*, September 2014.
8. Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. *arXiv preprint arXiv:1311.0243*, 2013.
9. Mike Hearn and Jeremy Spilman. Bitcoin contracts. `https://en.bitcoin.it/wiki/Contracts`. [Online; accessed May 2015].
10. G.O. Karame, E. Androulaki, and S. Capkun. Two Bitcoins at the Price of One? Double-Spending Attacks on Fast Payments in Bitcoin. In *Proc. of Conference on Computer and Communication Security*, 2012.
11. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. `https://bitcoin.org/bitcoin.pdf`. [Online; accessed March 26, 2014].
12. Joseph Poon and Thaddeus Dryja. The bitcoin lightning network.
13. Meni Rosenfeld. Analysis of hashrate-based double spending. *arXiv preprint arXiv:1402.2009*, 2014.
14. Yonatan Sompolinsky and Aviv Zohar. Accelerating bitcoin's transaction processing.
15. Pieter Wuille. BIP 0062: Dealing with Malleability. `https://github.com/bitcoin/bips`, 2014. [Online; accessed March 10th, 2014].