Preventing transaction delays with a Lightning routing service, for preventing abuse in a Lightning-based peer-to-peer exchange

C. J. Plooy (Bitonic)

22 may 2018

Contents

1 Introduction

1.1 The Lightning network

The Lightning network[1] forms a layer on top of a crypto-currency, where participants create *microtransaction channels*, which together form a network that allows payments to be routed, over multiple hops if necessary, from one participant to another. On a route A-B-C-D, A and B share a channel, B and C share a channel and C and D share a channel; a transaction from A to D is performed by updating the channel balances such that B receives funds from A, C receives funds from B and D receives funds from C. The amounts of funds transferred is equal, except (optionally) for a small fee charged by the intermediaries B and C; the net result (except for the fees) is that D receives the funds sent by A.

An important feature of the Lightning network is that the payer and the payee do not have to trust the intermediate nodes: transactions are set up in such a way that intermediate nodes can not receive incoming funds without also sending out outgoing funds. This is accomplished with HTLCs (Hash Time Locked Contracts): smart contracts (contracts written in the scripting language of the underlying crypto-currency) which specify that the receiving party can access the locked-in funds, but only if the pre-image of a certain hash is provided, and the sending party can take the funds back, but only after a certain (absolute) time. In the above example, HTLCs are used in the following sequence:

- Funds from A are locked in the A-B channel (hash: to B or timeout (3T): to A)
- Funds from B are locked in the B-C channel (hash: to C or timeout (2T): to B)
- Funds from C are locked in the C-D channel (hash: to D or timeout (1T): to C)
- Funds are released to D in the C-D channel after D sends the pre-image to C
- Funds are released to C in the B-C channel after C sends the pre-image to B
- Funds are released to B in the A-B channel after B sends the pre-image to A

Time-outs (1T, 2T, 3T) are selected such that each intermediate node is guaranteed to have some time, after receiving the preimage just before the time-out on the outgoing funds, to claim the incoming funds before their time-out expires.

1.2 Exchanging crypto-currencies over the Lightning network

It is not necessary for all channels in the Lightning network to consist of the same (crypto-currency) asset. It is even possible for different channels of the same node to be based on different underlying blockchains, provided that the time-out difference between HTLCs of transactions routed between the two is large enough to sufficiently reduce risks created by variability in block creation rates in the two blockchains.

If a node accepts an incoming transaction on one channel that is routed to another channel with a different asset, that node ends up receiving one asset and sending another asset. In other words: that node ends up performing an *exchange*. In the Lightning network, the payer initiates the payment, and determines the route and the amounts on all channels. Therefore, the payer also specifies the effective exchange rate of the transaction. Of course, an intermediate node can always choose to refuse forwarding a transaction if the exchange rate is unacceptable. This is equivalent to refusing a transaction because it pays insufficient fees to the intermediate node, or because the HTLC time-out difference for the intermediate node is insufficient.

An exchange transaction can include multiple exchanging parties: for instance, an ETH-holding customer can buy goods at a BTC-accepting shop with a route an ETH/LTC exchange and a LTC/BTC exchange. For simplicity, we will only consider routes that contain two exchanging parties. On Lightning, this ends up being a payment-to-self route, with one asset being sent from A to B and the other asset being sent from B to A.

2 The market

It is assumed an information exchange mechanism is possible where Lightning nodes can publish exchange offers, to buy or sell one asset in exchange for another asset. Such offers are informal and not cryptographically binding in any way. This is similar to the gossip protocol of the Lightning network, where nodes publish offers to forward payments at a certain fee rate. In both cases, there can be legitimate reasons to refuse payments, even if they conform to the terms and conditions of an earlier offer: for instance, channels might be saturated because of earlier transactions, or the exchange offer was already fully executed by earlier transactions. False offers can be identified relatively quickly by trying to route a payment, and observing it being canceled.

3 The delay attack

An exchange transaction has two exchanging parties: one offer-making node (OM) that placed an offer, and one offer-taking node (OT) that accepts the offer by making a route from OT to OM using one asset (X), and then back to OT using the other asset (Y). Both parties have a step in the Lightning protocol where they can delay the decision on whether to execute or cancel the transaction, up to the time specified in a HTLC time-out. These time-outs can be quite large, because they have to allow operators of intermediate nodes to manually repair/replace anything (including hardware and internet connections) that might break in the middle of ongoing transactions. In the mean time, the exchange rate between X and Y will be changed, either in the advantage of OT, or in the advantage of OM. In a route OT -(X)-> OM -(Y)-> OT, they can take advantage of creating delays in the following ways:

- After incoming X are locked in, OM can delay locking in outgoing Y. Just before OT's incoming HTLC time-out is expected, OM decides:
  - The value of X has increased w.r.t. Y -> lock in the outgoing Y.
  - The value of X has decreased w.r.t. Y -> cancel the transaction.

- After incoming Y are locked in, OT can delay claiming the incoming Y. Just before OT's incoming HTLC time-out is expected, OT decides:
  - The value of X has increased w.r.t. Y -> cancel the transaction.
  - The value of X has decreased w.r.t. Y -> claim the incoming Y.

Assuming both parties have the same impression of the X/Y exchange rate, if they both attack each other, they will always end up with a canceled transaction. With one attacking and one "honest" (non-attacking) party, the honest party will experience a long delay in transactions, have about half[1] of the transactions canceled, and receive sub-optimal exchange rates for the transactions that do succeed. It is therefore desirable to reduce or eliminate the opportunity for this attack.

4 First countermeasures

The offer-taking party can protect itself against the offering party in various ways. The offer-taking party might simply tell the offer-making party that it will immediately cancel any transaction that doesn't get locked in within a very small time window. This is a realistic threat, since, in the case where the offer-making benefits from delaying and then locking the transaction, the offer-taking party benefits from canceling. The offer-making party might then conclude that trying the abuse simply isn't worth it.

The offer-taking party can also protect itself by splitting up the transaction into small chunks. If one chunk is delayed, it can simply stop trusting the offer-making party, and choose to send the other chunks through other offers, if available at sufficiently attractive prices. Splitting up transactions is recommended anyway, since Lightning works much better for small transactions than for large ones.

Similarly, the offer-making party can protect itself by only accepting transactions in small chunks; the chunk size to be specified in the offer. Larger incoming transactions can simply be canceled immediately. This does not help in any way unless the offer-making party can somehow see who is performing transactions: otherwise, a single malicious party can simultaneously create many chunk-sized transactions, rendering the chunk size limit meaningless. Lightning provides no native way for the offer-making party to check the identity of the offer-taking party, but the offer-taking party might sign the transaction hash in advance to the offer-making party, to connect the transaction to a (public key) identity. Again, this doesn't help anything unless there is some kind of PKI and/or reputation system in place. With completely unverified keys, an attacker can simply generate and use a new key pair for every new chunk.

With these measures in place, the offer-taking party might be sufficiently protected, but protection for the offer-making party is quite limited. This might

[1]The honest node may change this ratio: for instance, with an exchange rate that is already significantly better for the attacker than the market exchange rate from the start is less likely to be canceled by the attacker.

discourage market participants from creating offers. In the absence of extra measures, there will probably be a price premium on offers; effectively, a large spread between buy and sell offers.

5 The routing service

The solution proposed here is to let a neutral route-making third party (RM) do the routing. So, instead of a route OT -(X)-> OM -(Y)-> OT, a route RM -(X)-> OM -(Y)-> OT -(X)-> RM is made. Alternatively, if RM prefers to deal wit asset Y, a route RM -(Y)-> OT -(X)-> OM -(Y)-> RM is made. OM and/or OT can add negative fees to pay RM for the service. Both exchanging parties (OT and OM) must trust the routing party to always cancel the transaction if it is delayed for a significant amount of time (say, more than a couple of seconds[2]). RM would not directly benefit from violating this rule, but it might cooperate with OT to the disadvantage of OM, or cooperate with OM to the disadvantage of OT. In a sense, the trust put in RM is comparable to the trust put in a 2-of-3 multisig escrow party with regards to the exchange rate risk.

5.1 The protocol

Since the routing service has to be trusted to some degree, it has to have a long-lived identity which can build up reputation. In this protocol description, it is assumed this identity corresponds to a known Lightning node ID of the routing service, so the exchanging parties know the node ID of the routing service.

The following names will be used for the three parties:

- RM: Route Maker: the routing service provider.
- OM: Offer Maker: the party making the published exchange offer. Is equal to either F or S.
- OT: Offer Taker: the party accepting the published exchange offer. Is equal to either F or S.
- F: First: the party receiving funds from RM, and sending funds to S.
- S: Second: the party receiving funds from F, and sending funds to RM.

The protocol is as follows:

1. OT contacts RM and requests a transaction hash. RM makes a transaction hash, signs it, and returns the signed hash to OT. Signing can for instance be done with the key corresponding to RM's node ID.

[2]This time-out must be large enough to let regular Lightning transaction execution fit reliably within the time-out. A time-out of seconds means there might still be room for high-frequency traders to game the system, but for regular users this should not be a major concern.

2. OT contacts OM and gives it RM's identity, the to-be-traded amounts and the signed transaction hash. OM checks whether the received data corresponds to an open offer, and ACKs.

3. (Optionally) using a rendezvous point R, S makes a route R - S - RM and hands over this Sphinx-encrypted route to F. The incoming amount of RM must equal what was agreed between OT and OM; the outgoing amount of R must equal what was agreed between OT and OM minus R's transaction fee. So, S pays the transaction fee on the route part it made.

   (a) In case S is OT, RM's fees must be added to the incoming amount of RM, so that OT pays RM's fee.

4. F extends this route to form C - F - R - S - C. The incoming amount of R must equal what was agreed between OT and OM; the outgoing amount of RM must equal what was agreed between OT and OM. So, F pays the transaction fee on the route part it made.

   (a) In case F is OT, RM's fees must be subtracted from the outgoing amount of RM, so that OT pays RM's fee.

5. F contacts RM, and hands over this route, as part of a request to perform the transaction.

6. RM looks at the transaction hash in the (non-encrypted) first hop of the route, and checks this is one of its own (not previously unused) transaction hashes, for which it has a preimage. RM starts the transaction.

7. Once the transaction locking reaches the incoming side of F, F checks whether the hash, incoming and outgoing amounts are as expected. If not, or if the transaction is late, it cancels the transaction.

8. Once the transaction locking reaches the incoming side of S, S checks whether the hash, incoming and outgoing amounts are as expected. If not, or if the transaction is late, it cancels the transaction.

9. Once the transaction locking reaches the incoming side of RM, RM checks whether the incoming amount equals the outgoing amount plus sufficient fees. If not, or if the transaction is late, it cancels the transaction.

10. Otherwise, RM reveals the preimage, kicking off the process of settling the transaction.

5.2 Properties of the protocol

1. None of the participants can steal any funds from each other. Either the transaction completes as agreed, or it is canceled.

2. RM can perform the delay attack, but it can only benefit if it cooperates with either F or S.

3. Neither F nor S can perform a delay attack during the locking phase, since then RM will unconditionally cancel the transaction.

4. Neither F nor S can perform a delay attack during the settling phase: the delaying party is already forced to pay, and can only choose between being paid or not being paid. Being paid is always the most beneficial option, and delaying this choice does not make it more profitable.

5. Both F and S know the route ends at RM, because they received a transaction hash signed by RM: a route that does not (effectively[3]) end at RM will never succeed with an RM-supplied transaction hash. Knowing the route ends at RM equals knowing that only RM can be free to choose whether or not the transaction succeeds.

6. It is possible to perform this kind of exchange without explicit multi-asset or multi-blockchain support in the Lightning software. F and S could run different nodes for the different assets, and there would be no need to do cross-asset channel advertizing in the Lightning gossip protocol. The only things needed from the Lightning software would be to allow the higher-level exchange software to catch and accept specially crafted routing requests that would otherwise be unrecognized by the Lightning software, and to send out transactions with customized route and transaction hash. The software of RM must be able to do a payment-to-self with a customized route, and to cancel incoming transactions after a very small time-out window since the transaction was initiated.

7. The node IDs of F and S can remain unknown to each other and to RM. The node ID of RM has to be known to F and S. However, there has to be direct communication between the three parties. For participants that wish to stay anonymous, this might for instance be accomplished through TOR.

5.3 A variation with less direct communication

To reduce the need for direct communication between participants, the following variation could be used:

1. OM contacts RM and requests a transaction hash. RM makes a transaction hash, signs it, and returns the signed hash to OM. Signing can for instance be done with the key corresponding to RM's node ID.

2. OM includes its own node ID and the signed transaction hash in the exchange offer.

3. OT makes a route RM - F - S - RM. The ratio between incoming and outgoing amounts for OM must be at least as good for OM as specified in

[3]It is possible for S to construct a route that continues beyond RM, but that has no consequences.

the offer. The incoming amount of RM must equal its outgoing amount, plus RM's fees. These requirements mean OT has to pay the fees, which in practice often translates into a slightly worse effective exchange rate for OT than specified in the offer.

4. OT contacts RM, and hands over this route and other data needed by RM.
5. RM continues as in the original protocol.

In this variation, no direct communication between both exchanging participants is needed, but direct communication with the routing server still exists. This variation has the following drawbacks compared to the original protocol:

- OT needs to know the node ID of OM; in fact, it needs to be present in the offer, so anybody can see it and associate it with a certain pair of assets, amounts and exchange rate. This is a reduction of privacy.
- The transaction hash needs to be present in the offer, so anybody, including RM and including intermediate nodes in the route, can see it and associate it with a certain pair of assets, amounts and exchange rate. This is a reduction of privacy.
- RM needs to issue potentially long-lived transaction hashes, and have a way to store the corresponding preimages. This is probably not a big problem though:
  - All preimages might be generated with some deterministic generator based on a single seed secret, similar to HD wallets. This should minimize storage requirements.
  - RM might set an expiry date on transaction hashes it issued, and refuse to create transactions that use an expired hash.

6 Conclusion

The system described here is not perfect, but when it comes to developing decentralized and trust-free alternatives to exchange services, it is an improvement. Compared to a regular exchange service, which has control over customers' funds, the routing service cannot steal from its customers, it cannot lose customers' funds in case of a hack, and unless the service provider decide to add restrictions on who / when to serve, it doesn't have to know any identity information about its customers, or even what asset is being traded between them, at what exchange rate.

7 Future work

A protocol has to be developed for publishing exchange offers. A peer-to-peer protocol is preferred to avoid a central point of control, but it may add undesired latency. The protocol, and especially peer behavior, has to be designed with some care, to prevent participants from being able to block the flow of information to other participants. It is to be determined whether such a decentralized market will suffer significantly from a flood of fake offers, and whether there are effective counter-measures.

Exchange offers are in a sense the same as Lightning routing offers, except they are cross-asset. It might make sense to publish exchange offers through Lightning's gossip network, but this is in no way necessary. In the version of the protocol described here, the node ID of the offering party can remain hidden; this reduces the similarity with Lightning routing offers, and also makes it harder to verify how serious an offer is (unlike a Lightning node, it is not fixed to on-chain assets). This could be an an argument to keep the two protocols separate, and/or to make this exchange protocol less anonymous (for instance, with a variation where the offer taker knows the offer maker node ID and makes the full route).

Information about available route maker services also needs to be exchanged. However, since trust is involved, it is probably best to not automatically find these services; instead, the user should manually search for parties it considers reliable, and add these to the configuration of its software.

References

[1] Joseph Poon and Tadge Dryja. Lightning network, 2015.