Modelling RFC003 Paramters

Lloyd Fournier, Lucas Soriano

March 2019

1 Introduction

COMIT RFC003 [1] defines a basic Hash Time Lock Contract (HTLC) based atomic swap. In order for the protocol to be secure, the HTLC expiry times must be set such that the parties have enough time to deploy and redeem the HTLCs. If they're not long enough the parties may risk losing funds or aborting the trade because they don't have enough time to safely continue the protocol. If they are too long, parties have their assets locked in HTLCs needlessly.

This documents seeks to create a simple model by which we can calculate the expiry times to make the protocol secure and practical. Note that it's currently incomplete and doesn't include the numerical methods actually required to calculate the expiry times. So far, It only includes expressions for the time taken to redeem β-`HTLC` and α-`HTLC` in terms of the model's parameters.

2 Beta Expiry

First we model `beta_expiry`, E_β, as it is the shorter of the two and is a parameter to calculating `alpha_expiry`. There are two ways in which `beta_expiry` has an effect on the protocol:

- How long Alice will (at least) have to redeem `beta_asset` if Bob funds.
- How long Bob will have to wait to be able to refund `beta_asset` if Alice does not redeem it.

Consequently, it would be ideal to find a value of `beta_expiry` that lets Alice redeem `beta_asset` with high probability, allowing the protocol to continue, while also ensuring Bob doesn't have to wait too long to be able to be refunded if it comes to that.

2.1 Parameters

1. T_A^α, T_A^β: the upper bound on the time Alice takes to confirm a transaction on `alpha_ledger` and `beta_ledger` respectively.

2. k_A^β: the number of additional confirmations Alice requires before she considers a transaction on `beta_ledger` to be permanent.

3. p_β: the required probability that Alice has enough time to redeem the β-`HTLC`, i.e. Alice will not abort due to insufficient time.

4. T_B^β: the upper bound on the time Bob takes to confirm a transaction on `beta_ledger`.

5. k_B^α: the number of additional confirmations Bob requires before he considers a transaction on `alpha_ledger` to be permanent.

6. $\lambda_\alpha, \lambda_\beta$: the *rate parameter* if `alpha_ledger`'s (resp. `beta_ledger`) block time is modelled as an exponential distribution [2], i.e. if Bitcoin is the `alpha_ledger` then $\lambda_\alpha = \frac{1}{10}$ (if we assume Bitcoin has a 10 minute block time).

2.2 Model

The events that must happen for Alice to safely and successfully redeem β-`HTLC` are:

1. Alice deploys the α-`HTLC` within T_A^α.

2. When α-`HTLC` has k_B^α additional confirmations, Bob deploys β-`HTLC` within T_B^β.

3. When β-`HTLC` has k_A^β additional confirmations, Alice redeems β-`HTLC` within T_A^β.

4. Alice must then get k_A^β additional confirmations on her redeem transaction (this is imprecise, see Limitations).

We may model waiting for k additional confirmations as the sum of k mutually independent exponential random variables with the same rate parameter λ. This is equivalent to an Erlang [3] distribution with shape k and rate λ.

We can model the time it takes for all these events to take place sequentially as a sum of random variables:

$$\begin{aligned} T_{\beta\texttt{-redeem}} = T_A^\alpha + \text{Erlang}\Big(k_B^\alpha, \lambda_\alpha\Big) + \\ T_B^\beta + \text{Erlang}\Big(k_A^\beta, \lambda_\beta\Big) + \\ T_A^\beta + \text{Erlang}\Big(k_A^\beta, \lambda_\beta\Big). \end{aligned}$$

Therefore, we should calculate E_β such that $E_\beta > T_{\beta\texttt{-redeem}}$ with probability p_β.

3 Alpha Expiry

There are two ways in which `alpha_expiry`, E_α, has an effect on the protocol:

- How long Bob will (at least) have to redeem `alpha_asset` if Alice redeems `beta_asset`.
- How long Alice will have to wait to be able to refund `alpha_asset` if the protocol isn't carried out to completion.

It would therefore be ideal to find a value of `alpha_expiry` that lets Bob redeem `alpha_asset` with high probability while also ensuring Alice doesn't have to wait too long to be able to be refunded if it comes to that.

3.1 Parameters

- T_B^α: the upper bound on the time Bob takes to confirm transactions on `alpha_ledger`.
- k_B^β: the number of additional confirmations Bob requires before he considers a transaction on `alpha_ledger` to be permanent.
- λ_α: the *rate parameter* if `alpha_ledger`'s block time is modelled as an exponential distribution.
- E_β: the value of `beta_expiry`, previously calculated.
- p_α: the required probability that Bob has enough time to redeem the α-`HTLC`. i.e. Bob doesn't lose his asset while failing to obtain the other.

3.2 Model

For Bob to safely redeem `alpha_asset` from α-`HTLC` he must have enough time after Alice has redeemed β-`HTLC` before E_β. To guarantee this we assume that Alice redeems at exactly E_β, so the time Bob has is $\Delta E = E_\alpha - E_\beta$. During ΔE the following events must happen with probability p^α:

1. Bob redeems α-`HTLC` within T_B^α.
2. Bob gets k_B^α additional confirmations on his redeem transaction (this is imprecise).

As in the Beta Expiry model, we model the time it takes to get k additional confirmations on a transaction as an Erlang random variable. We can model the time it takes for all these events to take place sequentially as a sum of random variables:

$$T_{\alpha\texttt{-redeem}} = E_\beta + T_B^\alpha + \text{Erlang}\left(k_B^\alpha, \lambda_\alpha\right).$$

Therefore, we should calculate E_α such that $E_\alpha > T_{\alpha\texttt{-redeem}}$ with probability p_α.

4 Limitations

4.1 Confirmation times

We don't try to explicitly model the time Alice and Bob take to confirm transactions, but rather take the upper bound on this time as parameters for each ledger/party pair, which are then used as constant random variables. We could try to model them using other distributions, but confirmation times are complicated to model because they depend on factors such as the transaction fee, transaction propagation time and the relative fees of transactions in the model. Estimates for confirmation times may also be given by wallet software or services like: bitcoinfees.earn.com and ethgasstation.info.

A simplifying and practical assumption could be that Alice and Bob always get their transaction into the next block.

4.2 Number of confirmations needed

The k parameters used above represent the number of additional confirmations needed before a party considers a transaction permanent. We use an Erlang random variable to represent the time taken to get these confirmations. This makes sense when accounting for how much time an actor will wait from the time the transaction is confirmed before they act on the transaction.

We also use it to model how much time an actor will need to get that many confirmations on their redeem transaction so it is fully confirmed before the HTLC expiry. But this will only be enough **as long as their transaction is never removed from the longest chain during this time**. This is an illogical assumption, as the reason you want to have confirmations is in case of such an event (and you need to re-submit the transaction).

To come up with a better estimate we would need to estimate the probability of a fork which removes the transaction during the confirmation period.

References

[1] Basic HTLC Atomic Swap. `https://github.com/comit-network/RFCs/blob/master/RFC-003-SWAP-Basic.md`, 2019.

[2] Wikipedia contributors. Exponential distribution — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Exponential_distribution`, 2018.

[3] Wikipedia contributors. Erlang distribution — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Erlang_distribution`, 2018.