Payment Channels Overview

Orfeas Stefanos Thyfronitis Litos

University of Edinburgh
`o.thyfronitis@ed.ac.uk`

Abstract. This is an overview of the existing literature on virtual payment channels. Lightning [1], Perun [2] and TeeChan [3] are considered.

1 Introduction

Virtual payment channels are constructions that permit the secure exchange of assets between remote agents without the need for each transaction to be recorded in a global database. They are constructed in a way that either does not allow the agents to cheat (given appropriate assumptions), or give the opportunity to the cheated agents to report the latest valid state to a global database (i.e. blockchain) and reclaim their assets.

2 Lightning Network

This construction is the first to achieve a functional model for payment channels. It is designed for bitcoin and requires some new opcodes and removing the malleability of transactions to function properly [1].

1 Simple two-party channel

The basic construction is as follows. Suppose that *Alice* and *Bob* want to create a payment channel that contains 1 BTC consisting of 0.5 BTC from each party. To achieve this, they follow these steps (see also section 3.1.2 and Figure 4 in 3.3.2 in [1]):

1. Either party (say *Alice*) creates a "Funding" transaction (F) with an input of 0.5 BTC from her and 0.5 BTC from *Bob*, and a 2-of-$\{Alice, Bob\}$ multisig as output; she then sends F to *Bob*. This transaction is not yet signed nor broadcast. F needs to be signed by both parties to be valid.
2. *Alice* creates, signs and sends to *Bob* a "Commitment" transaction ($C1b$) that spends F and has the following outputs:

(a) 0.5 BTC that can be spent by *Alice* immediately when $C1b$ is broadcast.

(b) 0.5 BTC that can be spent by either party, but *Bob* can spend it only after a specified amount of blocks (say n) have been mined on top of $C1b$, whereas *Alice* can spend it only if *Bob* provides her with a "Breach Remedy" transaction (explained later) signed by him. This output is called "Revocable Sequence Maturity Contract" (RSMC).

Furthermore, *Alice* creates, signs and sends a "Revocable Delivery" transaction ($RD1b$) that pays the first of the two outputs of $C1b$ to *Bob*, but will be accepted by the network if it is in the mempool only after n blocks have been mined on top of $C1b$.
Bob similarly creates, signs and sends $C1a$ and $RD1a$ to *Alice*.

3. After *Alice* receives the signed $C1a$ and $RD1a$ from *Bob*, she verifies that they are both valid and correctly spend F. Given that everything works out right, she signs F and sends it to *Bob*.
Observe that she is not running the risk of *Bob* refusing to cooperate in signing F and thus keeping her 0.5 BTC locked because she has the ability to sign and broadcast the (already signed by *Bob*) $C1a$ and $RD1a$ and thus get her money back n confirmations after $C1a$ is confirmed (that is when $RD1a$ is confirmed). Thus *Alice* need not trust *Bob* in any way.
Bob similarly verifies that $C1b$ and $RD1b$ have the correct structure, along with *Alice*'s signature on F. He then signs F and broadcasts it. Note that he does not have to trust *Alice* either.

After initially setting up the channel, *Alice* and *Bob* can update it as follows (see also section 3.3.4 and Figures 7, 8 in [1]):

1. Both *Alice* and *Bob* follow exactly the same steps as before to create $C2a$, $C2b$, $RD2a$ and $RD2b$; the only difference these transactions have to their counterparts from the previous state of the channel is that, instead of 0.5 BTC for each player, they contain the new agreed balance of the channel (e.g. 0.4 BTC for *Alice* and 0.6 BTC for *Bob*).
2. *Alice* creates, signs and sends to *Bob* a so-called "Breach Remedy" transaction ($BR1a$). This transaction lets *Bob* redeem the RSMC output of $C1a$ as soon as $C1a$ is broadcast. *Bob* similarly creates, signs and sends $BR1b$ to *Alice*.

Note that this effectively disincentivises *Alice* from ever broadcasting $C1a$, since in such case *Bob* will have a window of n blocks during which he can claim the entire sum in $C1a$, 1 BTC, for himself. *Alice* had better purge $C1a$ after $BR1a$ is sent to *Bob*. Similarly *Bob* is incentivised to refrain from ever broadcasting $C1b$.

This arrangement creates a situation where both players can be confident that the state of the channel is the one expressed by $C2a$, $C2b$, $RD2a$ and $RD2b$, thus they can assume that *Alice* has just paid *Bob* 0.1 BTC. No trust between the two players was needed all along. There are only two caveats: First, both players must periodically check the blockchain to ensure that the other party has not broadcast an old Commitment transaction. Second, in case of an uncooperative counterparty, one has to wait a prespecified amount of time before releasing their funds, which may be undesirable.

Thus, the necessary number of blocks mined on top of a Confirmation transaction for a subsequent Revocable Delivery to be valid (previously called n) must be carefully chosen in a way that does not lock up the funds for a long time in case of a dispute and at the same time does not require that the parties check the blockchain too often for a malicious broadcast of an already invalidated Commitment transaction.

Alice can outsource the task of the periodic check to a dedicated service by sending it all the previous Breach Remedy transactions. To incentivise the service to cooperate, *Alice* can pay a fee to it as an output of these transactions. Note that *Alice* does not need to trust the service, since the only thing it can do is to broadcast a Branch Remedy transaction that was created by *Alice*; she never discloses any of her private keys to it.

Finally, the parties can cooperatively close the channel without having to wait n blocks as follows: When both parties have agreed to closing the channel, *Alice* creates, signs and sends to *Bob* an "Exercise Settlement" transaction (ES) that spends the Funding transaction and has two simple outputs, each paying to the respective party the sum of the last agreed Commitment transaction. Following the previous example, this transaction would pay 0.4 BTC to *Alice* and 0.6 BTC to *Bob*. *Bob* can then also sign and broadcast the transaction to close the channel.

Once *Alice* has sent ES, she considers the channel as closed. If *Bob* does not broadcast ES, we have a dispute and she has to broadcast the latest Commitment transaction and wait for her funds to be unlocked.

3 Perun

The basic feature of Perun is the ability to create payment channels just like Lightning. Furthermore, it uses the Turing-completeness of Ethereum contracts to provide the option of so-called multistate channels; these channels can effectively support the creation of further virtual channels on top of them. Both kinds of channels require two moments of interaction with the Ethereum (or any other Turing-complete) blockchain: one at the creation and one when closing; the intermediate states of the channel need not be recorded on the blockchain.

As an example, of a virtual payment channel on top of multistate channels, let *Alice* and *Ingrid* have a multistate channel between them and let the same hold for *Ingrid* and *Bob*. Both multistate channels correspond to on-chain contracts. Then *Alice* and *Bob* can create a virtual payment channel with which *Ingrid* has to interact only at its creation; even in case that *Ingrid* later stops cooperating and does not help the other two parties to close the channel, they can both claim their money from their underlying multistate channel and eventually from the blockchain, thus *Ingrid* is required only for the virtual channel initiation.

One practical problem of this construction is that it does not scale well to virtual channels with many intermediaries. For *Alice* to create an $Alice \leftrightarrow Bob$ virtual payment channel, when the existing multistate channels are $Alice \Leftrightarrow Ingrid \Leftrightarrow Jane \Leftrightarrow Bob$, a virtual multistate channel specially crafted to support the desired virtual payment channel must be created between *Ingrid* and *Jane*. This means that when *Alice* requests from *Ingrid* to initiate the $Alice \leftrightarrow Bob$ channel through her and *Jane*, *Ingrid* has to establish a new channel with *Jane* before forwarding *Alice*'s request to *Jane*. Only then is the message from *Jane* to *Bob* ready. For *Alice* to be sure that the virtual payment channel is open, she has to wait for another message trip from *Bob* to *Jane*, then to *Ingrid* and eventually back to her. It seems that the amount of messages needed to create a virtual payment channel of length n is $\mathcal{O}\left(n^2\right)$. This is far from the ideal requirement of *Alice* being able to add a virtual channel with no interaction with any other party than *Bob*.

References

1. Poon J., Dryja T.: The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments
2. Dziembowski S., Eckey L., Faust S., Malinowski D.: PERUN: Virtual Payment Channels over Cryptographic Currencies. IACR: Cryptology ePrint Archive (2017)
3. Lind J., Eyal I., Pietzuch P., Sirer E. G.: Teechan: Payment Channels Using Trusted Execution Environments. ArXiv preprint arXiv:1612.07766 (2016)