Protocol for Recursive Virtual Channels

Aggelos Kiayias[1,2] and Orfeas Stefanos Thyfronitis Litos[1]

[1] University of Edinburgh
[2] IOHK
akiayias@inf.ed.ac.uk, o.thyfronitis@ed.ac.uk

Abstract. Protocol overview for Recursive Virtual Lightning-like payment channels on Bitcoin

Consider a sequence of parties $A_1, \dots, A_n$. We say that i is left of $i+1$ and $i+1$ is right of i. $\forall i \in \{2, \dots, n-1\}$, party A_i has a channel with A_{i-1} of total value $x_{i-1,i}$ and a channel with A_{i+1} of total value $x_{i,i+1}$. A_1 only has a channel with A_2 (of value $x_{1,2}$), likewise A_n only has a channel with A_{n-1} (of value $x_{n-1,n}$).

After following a specific protocol that does not involve any new on-chain transactions, each party holds off-chain a number of transactions and signatures that imply the existence of a new channel between A_1 and A_n with value x', funded by A_1. At a high level, these transactions are as follows:

- Each edge party has a transaction that consumes the funding output of its only channel and produces two outputs: one for the preexisting channel, where the left party has x' coins less and one that carries the x' coins for the virtual channel (read: the left party pays for the virtual channel). Call the latter "virtual output".
- Each intermediate party A_i has three types of transactions:
  - An "initiator" transaction, which consumes both its channel outputs and produces four: one for the left channel where the left party A_{i-1} has x' less coins, one for the right channel where A_i has x' less coins, one that pays A_i directly x' coins and one virtual output with x' coins.
  - Several "extend-interval" transactions which may be used if exactly one of the two adjacent parties has consumed the funding output of the shared channel. Wlog, assume that the party to the left has consumed the funding output $A_{i-1}A_i$ whereas the party to the right has not consumed A_iA_{i+1}. A_i's suitable extend-interval tx consumes A_iA_{i+1} and the virtual output produced by A_{i-1}'s transaction. In turn it produces one A_iA_{i+1} funding output where A_i has x' less coins, one output with x' coins for A_i and a new virtual output with x' coins.
  - Several "merge-intervals" transactions which can be used if both adjacent parties have consumed their respective funding output. The suitable merge-intervals tx consumes both virtual outputs from left and right and produces a new virtual output with x' coins and an output that pays A_i directly x' coins.

– Each party has one "commitment" transaction for each channel in which it takes part. This transaction can spend the latest funding output and produce one output for each party, each carrying the rightful amount. It also holds one "revocation" transaction per channel update which can be used to punish its counterparty if it publishes an old commitment transaction, by confiscating the entire channel value. To perform a payment, the two parties first create new commitment transactions with the new balance and then create revocation transactions for the old commitment transactions. This is in effect a simpler version of Lightning.

Q&A

– *Why each party holds many extend-interval and merge-intervals transactions?*

– When a series of neighbouring parties (an interval) have submitted their transactions, a party situated just beyond one edge of the interval can extend it by submitting its extend-interval transaction that corresponds to this specific interval. Therefore the party must have one extend-interval transaction for every possible interval which ends short of the party. The virtual output of this transaction specifies exactly the new, extended interval. Parties in the interval cannot make another move. For example in the case of 10 hops, if the spending condition "$all^8 \wedge$ `"4"`" is on-chain and unspent, it signifies that parties 4, 5, 6 and 7 have moved and that (if everyone is honest) parties 8 and 3 have not moved – it does not say anything about party 1, 2, 9 or 10.

  A merge-interval transaction can be published only if both funding outputs of the publishing party A have been consumed (either by initiator or by extend-interval transactions of its neighbours). The transactions of its two neighbours have one virtual output each, which the merge-interval transaction consumes and replaces with a single virtual output. Therefore eventually only one virtual output will remain, as intended. If instead the party at the other end of one of the two intervals publishes its transaction before A, it consumes one of the two aforementioned virtual outputs but produces a new one instead. This is however already planned for: A has another merge-interval transaction that consumes the new virtual output. A must hold one merge-interval transaction for every possible pair of surrounding intervals.

  Of note is that each intermediate party can only publish exactly one transaction. This transaction always generates exactly one new virtual output. If it is an initiator transaction, it does not consume a virtual output. If it is an extend-interval, it consumes one and if it is a merge-intervals it consumes two.

– *How can a malicious party diverge from honest execution? Does it affect balance security?*

– If a malicious intermediary sees an honest virtual output that it can spend with a valid merge-intervals transaction but the other virtual output needed as input for this merge-intervals transaction is not on-chain, it can create the missing virtual output using its own funds (all information needed for creating this is public) and publish its merge-intervals transaction. However, as the merge-intervals transaction has a virtual output with a wider interval, the same party cannot repeat the same trick. Since every new move widens the interval (it adds the mover to the previous interval), even if only one endpoint is honest, the attack cannot carry on for ever, therefore eventually the endpoint will be able to consume the virtual output as intended. Similar reasoning applies to extend-interval malicious transactions, where the malicious party fabricates the funding output. Regarding the case where a malicious party fabricates a virtual output and then publishes an extend-interval transaction that consumes this fabricated output and a valid funding output, we observe that the valid interval of the aforementioned virtual output include only parties that are not towards the direction of the honest counterparty. This means that the counterparty has the same view as if the malicious party had published honestly the extend-interval transaction, which causes no financial loss to the honest party. This fact does not change if more parties are malicious: the only possible difference for any honest party is the ability to spend more than one (extend-interval or merge-intervals) transactions and therefore gain more coins than if everyone were honest. Intuitively, any malicious party that fabricates a malicious output in order to spend an honest one just introduces more coins to the protocol in a way that does not allow it to gain value.

- *What if a malicious party publishes an old commitment transaction (i.e. consumes a funding output without using any of the initiator, extend-interval or merge-intervals txs)?*
- Its counterparty A_i won't be able to close honestly its other adjacent channel, but it will be able to punish the malicious party with the revocation transaction, thus confiscating all its funds. Therefore, to ensure no monetary loss is possible, A_i must always enforce that $x_{i-1,i,\text{right}} \leq x_{i,i+1,\text{right}}$ and $x_{i,i+1,\text{left}} \leq x_{i-1,i,\text{left}}$ (where $x_{i,j,\text{left/right}}$ is the value owned by the left-/right party of channel A_iA_j respectively). This balance check is performed on every payment and new virtual channel. NB: This is not too restrictive to not allow payments, but it is conjectured that this limitation can be lifted if an eltoo-based channel update method is used instead of the current, lightning-based method.
- *What about timelocks?*
- Virtual outputs can be consumed by extend-interval or merge-intervals as soon as they enter the ledger state, but if such a party does not publish its transaction after a while, then the virtual channel parties should be able to use this output as funding output for their virtual channel – this prevents griefing attacks. Therefore we need to put a timelocked spending condition on each virtual output spendable by the two parties that own the new virtual channel.

  Each such timelock should be long enough for each of the entitled intermediaries to have enough time to consume the virtual output, plus give a little leeway in case the party goes offline for a short period. Our construction allows the creation of "recursive" virtual channels, i.e. virtual channels that are built on top of other virtual channels. The funding outputs of the virtual channels exist off-chain and they need some time to reach the chain. The deeper an intermediary's channel is nested and the larger the number of hops that enabled this intermediary's channels, the longer has to be the timelock for the virtual outputs it is able consume.

– *What is the timelock value of a channel?*

–
$$t = \begin{cases} p + s & \text{if funding output on-chain,} \\ p + \sum\limits_{i=2}^{n-1} (s - 1 + t_i) & \text{else} \end{cases} , \qquad (1)$$

where t_i is the timelock of the i-th underlying intermediary party (whose timelock is the maximum of the timelocks of its two channels), s is the upper bound of η as in Lemma 7.19 of [1] and we arbitrarily choose $p = 3$ globally. This arises as the worst case delay, where a virtual channel owner submits its transaction and then every intermediary submits its extend-interval transaction at the latest possible moment, one after the other.

– *What is the protocol followed by channel parties to establish the necessary keys and signatures for the virtual channel transactions?*

– At a high level, the protocol consists of four roundtrips, each starting from the virtual channel funder to the first intermediary and then from each intermediary to the next, up to the virtual channel fundee. The first roundtrip is for key and timelocks distribution, where each party obtains all the necessary keys for all its required transactions, in the second roundtrip all signatures except for those needed to consume funding outputs are distributed, in the third roundtrip the parties exchange signatures that consume the funding outputs and finally in the fourth roundtrip parties revoke their old states. This structure ensures that parties only commit to the new channel state only after they have locally all the signatures necessary to enforce this state unilaterally.

Off-chain transactions

Syntactic shorthands:

– Let $A_i A_j$ the 2-of-$\{A_i, A_j\}$ spending condition.
– Let all^i an n-of-$\{A_1, \ldots, A_n\}$ spending condition for which all parties except for A_i have circulated their signatures.
– If there is a transaction spending a funding output $A_i A_j$, then $A_i^- A_j$ may be a new funding output of this transaction in which A_i owns x' coins less than in $A_i A_j$. Respectively for $A_i A_j^-$.

- Note that annotations in superscripts carry information which does not appear in the bitcoin script of the output, but are helpful for understanding the wider protocol.
- Literal numbers are used in spending conditions with quotes and monospace font, e.g. `"1"`.[a] This is used to indicate one end of an interval of parties that have made their move.
- A timelock t is represented with $+t$. The timelock required by A_i in all virtual outputs A_i is able to spend is represented with t_i.
- Spending conditions may be combined with $\wedge$ and $\vee$.
- An input or output is written as (spending condition(s), value). Inputs are ordered from left to right as viewed from the channel path, in outputs we put first the funding ones, then the reimbursement, then the virtual output.

Precondition: A_i can unilaterally put on-chain the funding outputs $(A_{i-1}, A_i, x_{i-1,i})$, $(A_i, A_{i+1}, x_{i,i+1})$ in finite time. Transactions:

- Held by A_1 (1 “initiator” tx):
  inputs:
  - $((A_1A_2), x_{1,2})$

  outputs:
  - $(A_1^- A_2, x_{1,2} - x')$
  - $(((all^2 \wedge \texttt{"1"}) \vee (A_1A_n + t_2)), x')$
- Held by A_n (1 “initiator” tx):
  inputs:
  - $((A_{n-1}A_n), x_{n-1,n})$

  outputs:
  - $(A_{n-1}^- A_n, x_{n-1,n} - x')$
  - $(((all^{n-1} \wedge \texttt{"}n\texttt{"}) \vee (A_1A_n + t_{n-1})), x')$

  A_1 and A_n do not hold “extend-interval” or “merge-intervals” transactions.
- Held by $A_i, i \in \{2, \ldots, n-1\}$:
  - Initiator transaction (1 tx):
    inputs:
    * $((A_{i-1}A_i), x_{i-1,i})$
    * $((A_iA_{i+1}), x_{i,i+1})$

    outputs:
    * $(A_{i-1}^- A_i, x_{i-1,i} - x')$
    * $(A_i^- A_{i+1}, x_{i,i+1} - x')$
    * (A_i, x')
    * $(((\text{if } (i-1>1) \ all^{i-1} \wedge \texttt{"}i\texttt{"})$
      $\vee(\text{if } (i+1<n) \ all^{i+1} \wedge \texttt{"}i\texttt{"})$
      $\vee(\text{if } (i-1>1 \vee i+1<n) \ A_1A_n + \max(t_{i-1}, t_{i+1})$
      $\quad \text{else } A_1A_n)),$
      $x')$
  - Extend-interval transactions ($n - 3 + \chi_{i=2} + \chi_{i=n-1}$ txs):
    * If $i = 2$ (1 tx):
      inputs:
      · $(all^2 \wedge \texttt{"1"}, x')$
      · $(A_2A_3, x_{2,3})$

outputs:
  · $(A_2^- A_3, x_{2,3} - x')$
  · (A_2, x')
  · (if $(n > 3)$ $((all^3 \wedge \texttt{"2"}) \vee (A_1 A_n + t_3))$ else $A_1 A_n, x')$
- ∗ If $i = n - 1$ (1 tx):
  inputs:
  · $(A_{n-2} A_{n-1}, x_{n-2,n-1})$
  · $(all^{n-1} \wedge \texttt{"}n\texttt{"}, x')$
  outputs:
  · $(A_{n-2}^- A_{n-1}, x_{n-2,n-1} - x')$
  · (A_{n-1}, x')
  · (if $(n - 2 > 1)$ $((all^{n-2} \wedge \texttt{"}n-1\texttt{"}) \vee (A_1 A_n + t_{n-2}))$ else $A_1 A_n, x')$
- ∗ $\forall k \in \{2, \ldots, i-1\}$ $(i - 2$ txs):
  inputs:
  · $(all^i \wedge \texttt{"}k\texttt{"}, x')$
  · $(A_i A_{i+1}, x_{i,i+1})$
  outputs:
  · $(A_i^- A_{i+1}, x_{i,i+1} - x')$
  · (A_i, x')
  · (((if $(k - 1 > 1)$ $all^{k-1} \wedge \texttt{"}i\texttt{"}$)
  $\vee$(if $(i + 1 < n)$ $all^{i+1} \wedge \texttt{"}k\texttt{"}$)
  $\vee$(if $(k - 1 > 1 \vee i + 1 < n)$ $A_1 A_n + \max(t_{k-1}, t_{i+1})$
    else $A_1 A_n$)),
  $x')$
- ∗ $\forall k \in \{i+1, \ldots, n-1\}$ $(n - i - 1$ txs):
  inputs:
  · $(A_{i-1} A_i, x_{i-1,i})$
  · $(all^i \wedge \texttt{"}k\texttt{"}, x')$
  outputs:
  · $(A_{i-1}^- A_i, x_{i-1,i} - x')$
  · (A_i, x')
  · (((if $(i - 1 > 1)$ $all^{i-1} \wedge \texttt{"}k\texttt{"}$)
  $\vee$(if $(k + 1 < n)$ $all^{k+1} \wedge \texttt{"}i\texttt{"}$)
  $\vee$(if $(i - 1 > 1 \vee k + 1 < n)$ $A_1 A_n + \max(t_{i-1}, t_{k+1})$
    else $A_1 A_n$)),
  $x')$

• Merge-intervals transactions ($(i - m) \cdot (l - i)$ txs – m, l defined below):
If $i = 2$ let $m \leftarrow 1$ else $m \leftarrow 2$.
If $i = n - 1$ let $l \leftarrow n$ else $l \leftarrow n - 1$.
$\forall k_1 \in \{m, \ldots, i-1\}, \forall k_2 \in \{i+1, \ldots, l\}$:
inputs:
- ∗ $(all^i \wedge \texttt{"}k_1\texttt{"}, x')$
- ∗ $(all^i \wedge \texttt{"}k_2\texttt{"}, x')$

outputs:
- ∗ (A_i, x')
- ∗ (((if $(k_1 > 2)$ $all^{k_1-1} \wedge \texttt{"}\min(k_2, n-1)\texttt{"}$)
  $\vee$(if $(k_2 < n - 1)$ $all^{k_2+1} \wedge \texttt{"}\max(k_1, 2)\texttt{"}$)

```
$\vee$(if $(k_1 > 2 \vee k_2 < n-1)$ $A_1A_n + \max(t_{k_1-1}, t_{k_2+1})$
  else $A_1A_n$)),
$x'$)
```

[a] This is used to specify one end of the interval. In Bitcoin script, if the scriptPubKey (on the output) begins with `OP_1 OP_EQUALVERIFY`, then the scriptSig (on the input) is valid only if it begins with `OP_1`.

References

1. Badertscher C., Maurer U., Tschudi D., Zikas V.: Bitcoin as a transaction ledger: A composable treatment. In Annual International Cryptology Conference: pp. 324–356: Springer (2017)