

# Multi-client Partially Non-Interactive and Instantaneous One-way Payment Channel for Ethereum

August 25, 2018 – DRAFT

Thomas Shababi<sup>1</sup>, Joël Gugger<sup>1</sup>, and Daniel Lebrecht<sup>1</sup>

TrueLevel SA, Neuchâtel, Switzerland  
{tom, joel, d}@truelevel.io

**Abstract.** Ethereum is a distributed computing platform and operating system featuring smart contract functionality. Transactions are faster than other blockchain but not instant and each of them cost some “gas”. This gas is used to quantify the amount of fee to pay for computation.

**Keywords:** Crypto-currencies, Ethereum, Payment channels

## 1 Introduction

**Requirements Language** The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119 [1].

## 2 Multi-clients payment channels

Partially non-interactive multi-clients payment channels are composed of clients  $c \in \mathcal{C}$  and one provider  $\mathcal{P}$ .  $\mathcal{C}$  is the set of all clients registered in the multi-clients payment channel. Clients can send money to the provider through the channel but cannot receive through the channel.

A payment channel is by definition a structure composed of two layers of states. The first layer of states is registered to the blockchain, i.e. “on-chain” or contract states  $\mu \in M$ , and the second layer of states is kept “off-chain” between the participants and the provider, i.e. channel states  $\sigma \in \Sigma$ . States are sets of deltas. A delta represent an atomic change in a state. Delta changes  $\delta \in \Delta$  are used to transition from one state to another.

We denote a main channel  $\omega \in \Omega$  and sub-channels  $\omega_{\alpha-\beta} \in \Omega$ . The main channel is composed by the two layers previously defined as  $\omega = (\Sigma, M)$ .

Each client have a pair of states  $(\mu_c, \sigma_c)$  shared with the provider. Only the latest pair is relevant, for each transition the new state in  $\Sigma$  or  $M$  is kept. We denote  $\mu_c$  and  $\sigma_c$  the contract and channel states for a client  $c \in \mathcal{C}$ . Hereinafter

it is implied that all  $\mu, \sigma$  correspond to one specific and only one client at the same time, but we forget  $c$  to simplify the notation.

Transition from a state to another state are denoted with  $\rightarrow$ , e.g.  $\sigma \rightarrow \sigma'$  or  $\mu \rightarrow \mu'$ . Transitions in  $\Sigma$  depends on particular deltas in  $\Delta_\sigma$  noted  $\delta_\sigma$ . We denote a transition from  $\sigma, \sigma' \in \Sigma$  that depends on  $\delta_\sigma \in \Delta_\sigma$  as

$$\sigma \xrightarrow{\delta_\sigma} \sigma'$$

And transitions in  $M$  depending on particular deltas in  $\Delta_\mu$  noted  $\delta_\mu$  as

$$\mu \xrightarrow{\delta_\mu} \mu'$$

Transitivity law is applied on transitions. If it exists a transition from a state  $s$  to another state  $s'$  that depends on  $t$  and  $s'$  to  $s''$  depending on  $t'$ , then it exists a transition from  $s$  to  $s''$  as

$$s \xrightarrow{t} s' \wedge s' \xrightarrow{t'} s'' \implies s \xrightarrow{t+t'} s''$$

We define predecessor and successor in states  $\Sigma$  and  $M$ . If state  $s$  is a predecessor of  $s'$  we denote them as  $s \prec s'$ , or  $s'$  is a successor of  $s$  as  $s' \succ s$ . We denote the latest transition of a certain type as  $\lceil \xrightarrow{t} \rceil$ .

Layers are linked together, changes in contract layer affect the channel layer and the channel layer can be dictated, or applied, on the contract layer. We denote the application of  $\sigma$  on  $\mu$ , i.e. dictate a channel state on the contract layer, also as  $\sigma(\mu) = \mu'$ . The full transition rule for contract states is define as

$$\mu \xrightarrow{\delta_\mu \vee \sigma(\mu)} \mu'$$

and the full transition rule for channel states as

$$\sigma \xrightarrow{\delta_\sigma \vee \mu \rightarrow \mu'} \sigma'$$

We can notice that each transition in contract states  $M$  trigger a transition in channel states  $\Sigma$ . States are represented as sets of deltas  $\delta \in \Delta$ . We define how to compute the resulted state.

$$\begin{aligned} \mu \xrightarrow{\delta_\mu} \mu' : \quad & \mu' = \mu \cup \{\delta_\mu\} \\ \mu \xrightarrow{\sigma(\mu)} \mu' : \quad & \mu' = \mu \cup \sigma \\ \sigma \xrightarrow{\delta_\sigma} \sigma' : \quad & \sigma' = \sigma \cup \{\delta_\sigma\} \\ \sigma \xrightarrow{\mu \xrightarrow{\delta_\mu} \mu'} \sigma' : \quad & \sigma' = \sigma \cup \mu' \vee \sigma \cup \{\delta_\mu\} \\ \sigma \xrightarrow{\mu \xrightarrow{\sigma''(\mu)} \mu'} \sigma' : \quad & \sigma' = \mu' \end{aligned}$$

It is worth noting that the last rule  $\sigma \xrightarrow{\mu \xrightarrow{\sigma''(\mu)} \mu'} \sigma'$  has implication on the validity of deltas. Because the resulting channel state  $\sigma'$  is equal to the contract

state  $\mu'$ , all deltas  $\delta_\sigma$  applied between the dictated state  $\sigma''$  and the starting state  $\sigma$ , where  $\sigma'' \preceq \sigma$ , are discarded. If we have

$$\begin{aligned}\Sigma : \emptyset &\xrightarrow{\delta_\mu} \mu_1 \xrightarrow{\delta_\sigma} \sigma_2 \xrightarrow{\delta'_\sigma} \sigma_3 \xrightarrow{\mu_1 \xrightarrow{\sigma_2(\mu_1)} \mu_2} \sigma_4 \\ M : \emptyset &\xrightarrow{\delta_\mu} \mu_1 \xrightarrow{\sigma_2(\mu_1)} \mu_2\end{aligned}$$

with application validity of a channel state  $\sigma$  on a contract state  $\mu$  ruled by

$$\sigma(\mu) \iff \sigma \succ \sigma'' : \sigma' \mid \xrightarrow{\mu \xrightarrow{\sigma'''} \mu'} \sigma''$$

We see that all channel states based on  $\sigma_4$  will never contain  $\delta'_\sigma$  because the dictated channel state used is  $\sigma_2$  and not  $\sigma_3$ , then as soon as  $\sigma_2$  has been applied  $\sigma_3$  became invalid by the given rule.

$$\begin{aligned}\mu_1 &= \emptyset \cup \{\delta_\mu\} &= \{\delta_\mu\} \\ \sigma_1 &= \emptyset \cup \{\delta_\mu\} &= \{\delta_\mu\} \\ \sigma_2 &= \{\delta_\mu\} \cup \{\delta_\sigma\} &= \{\delta_\mu, \delta_\sigma\} \\ \sigma_3 &= \{\delta_\mu, \delta_\sigma\} \cup \{\delta'_\sigma\} &= \{\delta_\mu, \delta_\sigma, \delta'_\sigma\} \\ \mu_2 &= \{\delta_\mu\} \cup \{\delta_\mu, \delta_\sigma\} &= \{\delta_\mu, \delta_\sigma\} \\ \sigma_4 &= \{\delta_\mu, \delta_\sigma\} &= \{\delta_\mu, \delta_\sigma\}\end{aligned}$$

### 3 Building blocks

#### 3.1 Contract state variables

We define variables to access the contract state  $\mu \in M$  for a client  $c \in \mathcal{C}$ .

**Current validity index,  $I_\mu$**  For each client the current index  $I_\mu$  must be retrievable,  $\forall c \in \mathcal{C}, \exists I_\mu \in \mu \geq 0$ . The index for a client must start from 0.

**Current contract client balance,  $B_\mu$**  For each client the current contract balance amount must be retrievable,  $\forall c \in \mathcal{C}, \exists B_\mu \in \mu \geq 0$ . It is worth noting that this current contract balance is not the same as the current channel balance or, in other words, the channel remaining amount of a client.

**Current provider refund amount,  $R_{\mathcal{P}}$**  For each client the current provider refund amount must be retrievable,  $\forall c \in \mathcal{C}, \exists R_{\mathcal{P}} \in \mu$  such that  $R_{\mathcal{P}}$  is the provider owned amount at the moment of the request. The remaining money must go to the client.

### 3.2 Contract states

Each client  $c \in \mathcal{C}$  in the contract is defined by an on-chain state  $\mu_c \in M$ . States  $\mu \in M$  are represented as

$$\mu_c = (I_\mu, B_\mu, R_\mathcal{P})$$

### 3.3 Channel state variables

We define variables to represent: (i) the lifetime of a single channel in the multi-clients channel architecture, (ii) the total amount deposited for a client over the lifetime, (iii) the total amount own by the provider over the lifetime, and (iv) the minimal and full available amount for the client.

**Channel lifetime** It exists one lifetime and only one per element in  $\mathcal{C} \times \mathcal{P}$  and begins when the first deposit is made, i.e. one lifetime per client  $c \in \mathcal{C}$ .

**Last observed validity index,  $I_\sigma$**  The last observed validity index is a copy of contract validity index at the moment where the state is created.

**Last observed client balance,  $B_\sigma$**  Same as the validity index but the client balance.

**Total deposit,  $D$**  Total deposit of a client represents the total amount recieved from the begining of the lifetime. Each top up increases the total deposit.

**Total spent,  $S$**  Total amount sent to the provider by a client represents the sum of all payments since the beginning of the lifetime. Each payment increases the total spent.

**Client refunded amount,  $R$**  Total amount refunded to the client since the beginning of the lifetime. Each refund can increase the total client refunded amount.

**Minimal available amount,  $A_m$**  Minimal amount available for a client is computed with the latest channel state  $\sigma$ . Without quering the contract state  $\mu$  it is impossible to know if a bigger amount is now available. The minimal available amount is computed with

$$A_m = D - S - R$$

**Client available amount,  $A$**  The full amount available into a single channel for a client is computed with

$$A = D - S - R + B_\mu - B_\sigma, \quad B_\mu \geq A$$

It is worth noting that  $B_\mu - B_\sigma$  is added to the difference of total deposit and total spent in case of new on-chain delta changes.

### 3.4 Channel states

Each client  $c \in \mathcal{C}$  is defined by their channel state  $\sigma \in \Sigma$ . States  $\sigma \in \Sigma$  are composed of: (i) a validity index, (ii) the latest observed on-chain balance of the client, (iii) the total deposit of the client, (iv) the total owned by the provider, and (v) the client total refunded amount. We denote a state  $\sigma_c \in \Sigma$  as

$$\sigma_c = (I_\sigma, B_\sigma, D, S, R)$$

### 3.5 Modifying channel states and contract states with Deltas

Deltas  $\delta \in \Delta$  represent external channel capabilities of changing states  $\Sigma$  and contract states  $M$ . Along with directives, they encapsulate all the channel capabilities. In this proposed channel architecture capabilities are: send money through the channel from clients to provider, create and top up a channel, settle a channel, and ask and get a refund. In that list, external capabilities are sending money through the channel, creating a channel, and top up a channel.

**Payment,  $\delta_\sigma(a)$**  This delta is used to increase the channel balance of the provider  $O(\mathcal{P})$  of  $a$  amount from a client  $c$ . Payment deltas are sent by clients to the provider. For each payment an acquired settle directive  $d_s$  is generated by the client and sent to the provider.

**Top up,  $\delta_\mu(a)$**  This delta is used to increase the contract balance of a client by  $a$  amount. Top up deltas are sent by anyone for any specific channel, i.e. client.

Creating channels is a special case of top up, when a channel is top up but does not already exists, the channel is created. The first delta of all channel will always be a top up.

### 3.6 Dictate channel state on contract layer with Messages & Directives

We define in the previous chapter a transition in contract layer performed with a channel state noted  $\mu \xrightarrow{\sigma} \mu'$  or  $\sigma(\mu) = \mu'$ . These transitions are performed by the intermediary of messages  $m \in \mathcal{M}$  and directives  $d$ .

Messages  $m \in \mathcal{M}$  are an intermediary representation of channel states between the channel layer and contract layer. A message is related to one and only one element in  $\mathcal{C} \times \mathcal{P}$  for only one  $\sigma \in \Sigma$ . Messages are used to derive directives. Directives are the concrete representation of a state  $\sigma$  in the contract layer. The contract layer only “speak” and understand directives.

Thus we will use the message representation in the following chapter to describe contract transitions based on channel state application.

**Minimal message** A minimal message  $m \in \mathcal{M}$  between a client  $c$  and the provider  $\mathcal{P}$  is composed of four components: (i) a validity index, (ii) the latest observed balance, (iii) the adjusted total of deposit, and (iv) the total owned by the provider (total spent by the client). We denote a message  $m \in \mathcal{M}$  as

$$m = (I_m, B_\sigma, D_m, S)$$

**Message derivation** Messages are derived from channel states  $\Sigma$  as

$$\begin{aligned} \sigma &\Longrightarrow m \\ (I_\sigma, B_\sigma, D, S, R) &\Longrightarrow (I_\sigma + 1, B_\sigma, D - R, S) \end{aligned}$$

**Directives** One message can derive many directives. We define two base directives: (i) one used by the provider and (ii) one used by the client. It is possible to add as many directives as needed in the contract layer. We distinguish two types of directives: (i) self-generated directives, and (ii) acquired directives. Self-generated directives can be generated without external computation and are directly valid in the contract layer. Conversely acquired directives require external computation with the other party (can be client or provider).

Settling and refunding a channel are part of dictating a channel state on the contract layer, and then are performed with directives.

**Settle,  $d_s$**  Directive sent from the client to the provider to validate a payment  $\delta_\sigma(a)$ . The process of generating the directive can be summarized as

$$\forall \sigma \xrightarrow{\delta_\sigma(a)} \sigma', \exists d_s : \sigma' \Longrightarrow m \Longrightarrow d_s$$

**Refund,  $d_r$**  Self-generated directive by the client used to ask for a refund. The process of generating the directive can be summarized as

$$\forall \sigma, \exists d_r : \sigma \Longrightarrow m \Longrightarrow d_r$$

When a client asks for a refund, the provider has three choices: (i) doing nothing, (ii) resolve the refund with a matching  $d_r$ , or (iii) dispute the refund with a matching  $d_r$ . If the provider does nothing the client can take what he asked after some delay, otherwise the provider will close the request by resolve or dispute.

## 4 State transitions

### 4.1 Contract state, $\mu \rightarrow \mu'$

Contract states  $\mu \in M$  can transition from  $\mu$  to  $\mu'$  with a message  $m \in \mathcal{M}$  (i.e. directive) or contract state changes  $\delta_\mu \in \Delta$ .

**Settlement**  $(I_\mu, B_\mu, R_{\mathcal{P}}) \xrightarrow{m} (I_\mu + 1, B_\mu \downarrow, R_{\mathcal{P}})$  where

$$m = (I_m, B_\sigma, D_m, S) \mid I_m = I_\mu + 1$$

It is worth noting that with this rule it is not possible to settle a zero amount because the balance amount  $B_\mu$  must go down.  $O(\mathcal{P})$  is the amount owned by the provider at the settlement time. The provider balance is the current contract balance minus the client available funds. Computed as

$$\begin{aligned} O(\mathcal{P}) &= B_\mu - A \\ &= B_\mu - (D_m - S + B_\mu - B_\sigma) \end{aligned}$$

The new balance  $B_\mu \downarrow$  is the remaining funds

$$\begin{aligned} B_\mu \downarrow &= A \\ &= D_m - S + B_\mu - B_\sigma \end{aligned}$$

The new client balance is the current contract balance minus the amount due to the provider.

**Refund**  $(I_\mu, B_\mu, R_{\mathcal{P}}) \xrightarrow{m} (I_\mu + 1, B_\mu, R'_{\mathcal{P}})$  where

$$m = (I_m, B_\sigma, D_m, S) \mid I_m = I_\mu + 1$$

It is a full refund if the client available funds in the channel for the client are equal to the current contract balance and a partial refund if the available funds of the client are smaller than the balance

$$R'_{\mathcal{P}} = \begin{cases} B_\mu - A & \text{if } B_\mu > A \\ 0 & \text{if } B_\mu = A \end{cases}$$

**Dispute**  $(I_\mu, B_\mu, R_{\mathcal{P}}) \xrightarrow{m} (I_\mu, 0, 0)$  where

$$m = (I_m, B_\sigma, D_m, S) \mid I_m = I_\mu$$

If  $O(\mathcal{P}) > R_{\mathcal{P}}$  the provider won the dispute and  $O(\mathcal{P})$  is transfer to the provider plus some penalties, the rest is sent to the client.

**Resolve**  $(I_\mu, B_\mu, R_{\mathcal{P}}) \xrightarrow{m} (I_\mu, 0, 0)$  where

$$m = (I_m, B_\sigma, D_m, S) \mid I_m = I_\mu$$

If  $O(\mathcal{P}) = R_{\mathcal{P}}$  the provider accept the refund.  $O(\mathcal{P})$  is transfer to the provider and the rest is sent to the client immediately.

**Top up**  $(I_\mu, B_\mu, R_\mathcal{P}) \xrightarrow{\delta_\mu(a)} (I_\mu, B_\mu + a, R_\mathcal{P})$ . Top up increases the contract balance for a client  $c \in \mathcal{C}$ . Validity index  $I_\mu$  must not be incremented during the top up.

**Invalid settlement**  $(I_\mu, B_\mu, R_\mathcal{P}) \rightarrow (I_\mu, B_\mu \downarrow, R_\mathcal{P})$ . This transition is invalid because the balance is decreased without incrementing the validity index  $I_\mu$ . The current set of transactions must be invalidate after the settlement.

**Invalid top up**  $(I_\mu, B_\mu, R_\mathcal{P}) \rightarrow (I_\mu + 1, B_\mu \uparrow, R_\mathcal{P})$ . Increasing validity index  $I_\mu$  while increasing the contract balance invalidates the set of transaction  $I_\mu$ . The current set of transactions must be invalidated only during a settlement.

**Invalid validity index**  $(I_\mu, B_\mu, R_\mathcal{P}) \rightarrow (I_\mu \downarrow, -, -)$ . Decreasing the validity index  $I_\mu$  is always invalid. Validity index must not be decreased.

#### 4.2 Channel state, $\sigma \rightarrow \sigma'$

Channel state transitions  $(\sigma \xrightarrow{\delta_\sigma \vee \mu \rightarrow \mu'} \sigma')$  are triggered by delta changes  $\delta_\sigma \in \Delta$  or by transitions in  $M$ , i.e.  $\mu \rightarrow \mu'$ . The base channel state  $\sigma$  is computed as

$$\sigma = \begin{cases} \emptyset & \text{if no previous state exists} \\ (I_\sigma, B_\sigma, D, S, R) & \text{otherwise} \end{cases}$$

**Payment,  $\delta_\sigma(a)$** . Payments trigger transition in channel states from the base state  $\sigma$  to a destination state  $\sigma'$  noted  $\sigma \xrightarrow{\delta_\sigma(a)} \sigma'$ . The destination channel state is computed as

$$\sigma' = \begin{cases} \emptyset & \text{if } \sigma = \emptyset \\ (I_\mu, B_\mu, D, S + a, R) & \text{otherwise} \end{cases}$$

Payments cannot be accepted without a previous state. If  $\sigma = \emptyset$  then the channel is not initialized yet and must not accept any payments.

**Top up,  $\mu \xrightarrow{\delta_\mu(a)} \mu'$** . Each top up event modify the contract state  $\mu$ , then the channel state  $\sigma$  must be updated to keep track of the real balance. The destination channel state is computed as

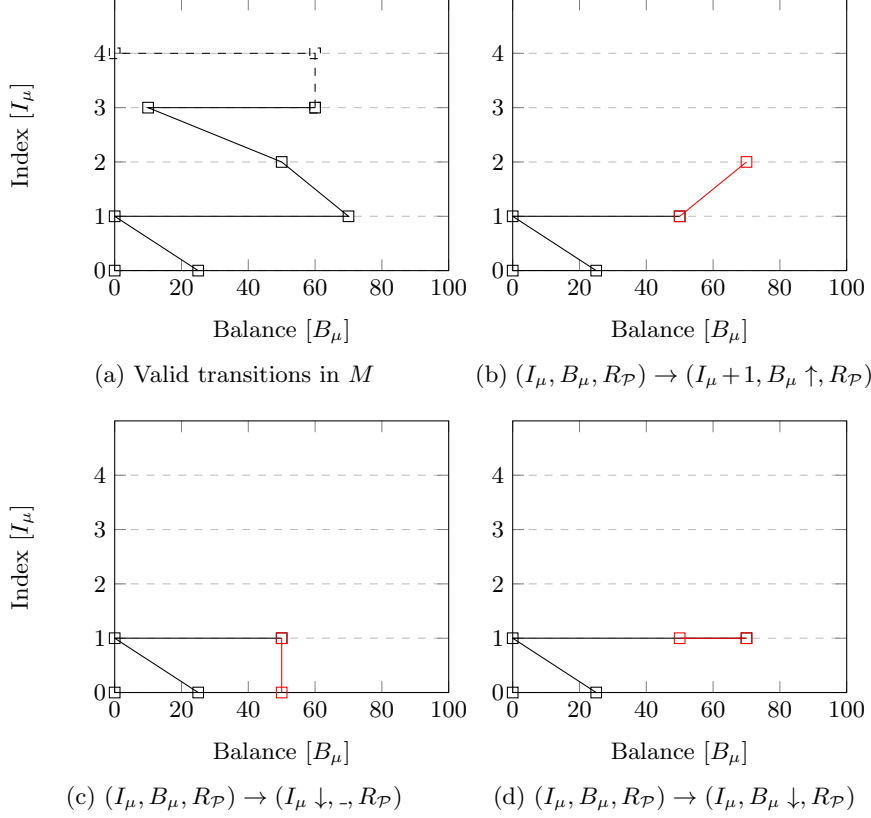
$$\sigma' = \begin{cases} (0, B_\mu, B_\mu, 0, 0) & \text{if } \sigma = \emptyset \\ (I_\sigma, B_\mu, D + B_\mu - B_\sigma, S, R) & \text{otherwise} \end{cases}$$

where

$$B_\mu - B_\sigma = a$$

It is worth noting that if no previous state exists the channel is initialized and the first state is created.





**Fig. 1.** State transitions in  $M$ , transitions in red are not allowed. Dashed transition are during refund process. The first step is the refund request, the second can be either claim, resolve, or dispute. Figure (c) show validity index decrementation, this is invalid no matter of the resulting balance amount. Figure (d) shows a decreasing balance, red line overflow the black line.

**Settle,  $\mu \xrightarrow{\sigma''} \mu'$ .** Settlements trigger transition in contract state  $\mu \rightarrow \mu'$  and then the channel state  $\sigma$  must be updated as

$$\sigma' = \begin{cases} \emptyset & \text{if } \sigma = \emptyset \\ (I_\mu, B_\mu, D, S, R) & \text{otherwise} \end{cases}$$

**Refund,  $\mu \xrightarrow{\sigma''} \mu' \mid R_{\mathcal{P}} \geq 0$ .** Clients can ask for refund and modify the contract state then the channel state must be updated as

$$\sigma' = \begin{cases} \emptyset & \text{if } \sigma = \emptyset \\ (I_\mu, B_\sigma, D, S, R) & \text{otherwise} \end{cases}$$

and the state of the channel must enter is “refund mode” until the refund is ended with either a dispute, resolve, or a claim. During this periode client must not send new payments to prevent the provider to successfully dispute the refund (even if updating the channel state to  $I_\sigma = I_\mu$  normlay prevent negotiating payment for  $I_m = I_\mu$ ).

**Dispute,  $\mu \xrightarrow{\sigma''} \mu'$ .** When request refunds are resolved with a dispute the provider can take what he owns during the dispute plus some penalties  $\rho$ .

$$\sigma' = \begin{cases} \emptyset & \text{if } \sigma = \emptyset \\ (I_\sigma, B_\mu, D, S + \rho, B_\mu - R_P - \rho) & \text{otherwise} \end{cases}$$

such that

$$D = S + R \wedge B_\mu = 0$$

**Resolve & Claim,  $\mu \xrightarrow{\sigma''} \mu'$ .** Provider can accept the request and resolve a refund request by submitting the settle directive  $d_s$  or a client can claim his funds after the delay. In both cases the contract state  $\mu$  changes, then the channel state must be updated as

$$\sigma' = \begin{cases} \emptyset & \text{if } \sigma = \emptyset \\ (I_\sigma, B_\mu, D, S, B_\mu - R_P) & \text{otherwise} \end{cases}$$

such that

$$D = S + R \wedge B_\mu = 0$$

## 5 Protocol

The protocol describes interactions between clients and the provider and messages negotiation. Protocol define the derivation of directives from messages.

**Client actions** A client can send money through the channel and refund his money after some time if he submitted the right message  $m$  corresponding to the lastest channel state  $\sigma$ .

**Provider actions** The provider can settle channels with his lastest message corresponding to the latest channel state. If a client requested a refund, the provider has the choice to dispute it or resolve it.

**Anyone actions** Anyone can top up a channel for himself or for another client.

	$\delta$	$\mu$	$m$	$\sigma$
	$\delta(a)$	$(I_\mu, B_\mu, R_\mathcal{P})$	$(I_m, B_\sigma, D_m, S)$	$(I_\sigma, B_\sigma, D, S, R)$
		$(0, 0, 0)$		$\emptyset$
top up	$\delta_\mu(10)$	$(0, 10, 0)$		$(0, 10, 10, 0, 0)$
payment	$\delta_\sigma(1)$		$(1, 10, 10, 1)$	$(0, 10, 10, 1, 0)$
payment	$\delta_\sigma(1)$		$(1, 10, 10, 2)$	$(0, 10, 10, 2, 0)$
settle		$(1, 8, 0)$		$(1, 8, 10, 2, 0)$
payment	$\delta_\sigma(2)$		$(2, 8, 10, 4)$	$(1, 8, 10, 4, 0)$
payment	$\delta_\sigma(2)$		$(2, 8, 10, 6)$	$(1, 8, 10, 6, 0)$
top up	$\delta_\mu(10)$	$(1, 18, 0)$		$(1, 18, 20, 6, 0)$
payment	$\delta_\sigma(1)$		$(2, 18, 20, 7)$	$(1, 18, 20, 7, 0)$
settle		$(2, 13, 0)$	$(3, 13, 20, 7)$	$(2, 13, 20, 7, 0)$
refund		$(3, 13, 0)$		$(3, 13, 20, 7, 0)$
claim		$(3, 0, 0)$		$(3, 0, 20, 7, 13)$

**Table 1.** State transitions during channel lifetime

### 5.1 Payment, $\delta_\sigma(a)$

Payment are negotiated with of one acquire directive for the provider signed by the client.

**Settle directive,  $d_s$**  Settle directive  $d_s$  allow the provider to claim immediately, i.e. without any delay, his owned amount for a specific channel. The client must not be able to applied a settle directive on behalf of the provider.

$$h = \text{keccak256}(\text{contract}, c, I_m, B_\sigma, D_m, S)$$

$$d_s = \text{sign}(\text{prefix}, h) \quad \text{with } c \text{ private key}$$

### 5.2 Refund

Opening a refund is performed with one directive self-generated by the client and signed by himself. The provider must not be able to create directives  $d_r$  nor applied a refund directive on behalf of the client.

**Refund directive,  $d_r$**  Refund directive  $d_r$  allow the client to claim all the remaining funds and transfer them after some delay if the provider is not able to dispute the claim nor did not resolve the request. The provider is not able to

	$\delta$	$\mu$	$m$	$\sigma$
	$\delta(a)$	$(I_\mu, B_\mu, R_{\mathcal{P}})$	$(I_m, B_\sigma, D_m, S)$	$(I_\sigma, B_\sigma, D, S, R)$
		$(0, 0, 0)$		$\emptyset$
top up	$\delta_\mu(3)$	$(0, 3, 0)$		$(0, 3, 3, 0, 0)$
top up	$\delta_\mu(2)$	$(0, 5, 0)$		$(0, 5, 5, 0, 0)$
payment	$\delta_\sigma(3)$		$(1, 5, 5, 3)$	$(0, 5, 5, 3, 0)$
settle		$(1, 2, 0)$		$(1, 2, 5, 3, 0)$
payment	$\delta_\sigma(1)$		$(2, 2, 5, 4)$	$(1, 2, 5, 4, 0)$
top up	$\delta_\mu(2)$	$(1, 4, 0)$		$(1, 4, 7, 4, 0)$
settle		$(2, 3, 0)$		$(2, 3, 7, 4, 0)$
payment	$\delta_\sigma(2)$		$(3, 3, 7, 6)$	$(2, 3, 7, 6, 0)$
refund		$(3, 3, 2)$		$(3, 3, 7, 6, 0)$
top up	$\delta_\mu(10)$	$(3, 13, 2)$		$(3, 13, 17, 6, 0)$
resolve		$(3, 0, 0)$		$(3, 0, 17, 6, 11)$
top up	$\delta_\mu(15)$	$(3, 15, 0)$		$(3, 15, 32, 6, 11)$
payment	$\delta_\sigma(7)$		$(4, 15, 21, 13)$	$(3, 15, 32, 13, 11)$
payment	$\delta_\sigma(5)$		$(4, 15, 21, 18)$	$(3, 15, 32, 18, 11)$
top up	$\delta_\mu(3)$	$(3, 18, 0)$		$(3, 15, 35, 18, 11)$
settle		$(4, 6, 0)$		$(4, 6, 35, 18, 11)$

**Table 2.** Settlement after top up and refund resolved

dispute the claim if the directive is generated from the message derived by the latest channel state in  $\Sigma$ .

$$h = \text{keccak256}(\text{contract}, c, I_m, O(\mathcal{P}))$$

$$d_r = \text{sign}(\text{prefix}, h) \quad \text{with } c \text{ private key}$$

with

$$O(\mathcal{P}) = B_\mu - A$$

$$= B_\mu - (D_m - S + B_\mu - B_\sigma)$$

A refund request can be closed in tree different maners

$$\text{close a refund by } \begin{cases} \text{dispute} & \text{if the provider is able to prove a lie} \\ \text{resolve} & \text{if the provider agrees} \\ \text{claim} & \text{otherwise by the client after delay} \end{cases}$$

### 5.3 Dispute refund

Disputes are won if the provider can prove that it owns a payment directive  $d_s$ , i.e. client agreed on a later channel state  $\sigma'$ , where

$$O(\mathcal{P})' > O(\mathcal{P}), \quad O(\mathcal{P})' \in d_s \wedge O(\mathcal{P}) \in d_r$$

with

$$\begin{aligned} O(\mathcal{P})' &= B_\mu - A \\ &= B_\mu - (D_m - S + B_\mu - B_\sigma) \end{aligned}$$

### 5.4 Resolve refund

Provider can choose to agree with the request and resolve it directly. By resolving the request the provider will get his remaining funds and send the client available funds  $A$  directly to the client. It is worth noting that the provider has no incentive to resolve request where  $S = 0$ .

### 5.5 Claim refund

The client can claim his funds after the delay if the provider has not disputed nor resolved the request.

## References

- [1] S. Bradner. *IETF RFC 2119: Key words for use in RFCs to Indicate Requirement Levels*. Tech. rep. 1997.