Nitro Protocol

Tom Close

December 12, 2018

1 Recap of ForceMove

The ForceMove protocol describes the message format and the supporting on-chain behaviour to enable generalized, n-party state channels on any blockchain that supports Turing-complete, general-purpose computation. Here we give a brief overview of the protocol to the level required to understand the rest of the paper. For a more comprehensive explanation please refer to [].

A ForceMove **state channel**, $\chi(P, L, k)$, is defined by an ordered set of participant addresses, $P = [p_0, ..., p_{n-1}]$, the address of an on-chain **game library**, L, and a nonce, k, which chosen by the first participant to make the channel's combination of properties unique. The **channel address** is calculated by taking the last 20 bytes of the `keccak256` hash of the channel properties.

`participants`	`address[]`	The addresses used to sign updates to the channel.
`gameLibrary`	`address`	The address of the gameLibrary, which defines the transition rules for this channel
`nonce`	`unit256`	Chosen to make the channel's address unique.
`turnNum`	`unit256`	Increments as new states are produced.
`balances`	`(address, uint256)[]`	Current *outcome* of the channel.
`isFinal`	`bool`	
`data`	`bytes`	
`v`	`uint8`	ECDSA signature of the above arguments by the moving participant.
`r`	`bytes32`	
`s`	`bytes32`	

Table 1: ForceMove state format

A ForceMove **state**, $\sigma^i_\chi(\beta, f, \delta)$, is specified by **turn number**, i, a set of **balances**, β, a boolean flag **finalized**, $f \in \{T, F\}$, and a chunk of unstructured **game data**, δ, that will be interpreted by the game library. The balances can be thought of as an ordered set of (`address`, `uint256`) pairs, which specify how any funds allocated to the channel should be distributed if the channel were to finalize in the current state.

In order for a state, σ^i_χ, to be valid it must be signed by participant, p_j, where $j = i\%n$ is the remainder mod n. This requirement specifies that participants in the channel must take turns when signing states.

The game library is responsible for defining a set of states and allowed transitions that in turn define the 'application' that will run inside the state channel. It does this by defining a single boolean function, $t_L(i, \beta, \delta, \beta', \delta') \to \{T, F\}$. This function is used to derive an overall boolean transition function, t, specifying whether a transition between two states is permitted under the rules of the protocol:

$$\begin{aligned} t(\sigma^i_\chi(\beta, f, \delta), \sigma^j_{\chi'}(\beta', f', \delta')) \Leftrightarrow & \chi = \chi' \wedge j = i + 1 \wedge \\ & [(\neg f \wedge \neg f' \wedge j \leq 2n \wedge \beta = \beta' \wedge \delta = \delta') \vee \\ & (\neg f \wedge \neg f' \wedge j > 2n \wedge t_L(n, \beta, \delta, \beta', \delta')) \vee \\ & (f' \wedge \beta = \beta' \wedge \delta' = 0)] \end{aligned}$$

In all transitions the channel properties must remain unchanged and the turn number must increment. There are then three different modes of operation. The first mode applies in the first $2n$ states (assuming none of these are finalized) and in this mode the balances and game data must remain unchanged. As we will see later, these states exist so that the channel can be funded safely. We refer to the first n states as the **pre-fund setup** states and the subsequent n as the **post-fund setup** states. The second mode applies to the 'normal' operation of the channel, when the game library is used to determine the allowed transitions. The final mode concerns the finalization of the channel: at any point the current participant can choose to exit the channel and lock in the balances in the current state. Once this happens the only allowed transitions are to additional finalized states. Because of this, we have no further use for the game data, δ, so can remove this from the state. Once a sequence of n finalized states have been produced the channel is considered closed. We call this sequence of n finalized states a **conclusion proof**.

1.0.1 On-chain operations

Adjudicator

We will use the notation $[\![.]\!]$

$$D_\chi(x)[\![\alpha_\chi(0)]\!] \to [\![\alpha_\chi(x)]\!]$$

$$W_A^{FM}(x)[\![\alpha_\chi(x+y)\beta_\chi(A:x+z,...)]\!] \to [\![\alpha_\chi(y)\beta_\chi(A:z,...)]\!]$$

$$\Theta(\tau+\epsilon)[\![\kappa(\tau,\sigma_\chi(\beta_\chi,\delta))]\!] \to [\![\beta_\chi\kappa_\chi(\bot)]\!]$$

$$FM(\tau,\sigma^i,...,\sigma^{i+n-1})[\![\kappa_\chi(\top)]\!] \to [\![\kappa(\tau+\eta,\sigma^{i+n-1})]\!]$$

$$R(\tau',\sigma^{i+1})[\![\kappa(\tau,\sigma^i)]\!] \to [\![\kappa_\chi(\top)]\!]$$

$$C(\tau,\sigma^i,...,\sigma^{i+n-1}(\beta))[\![\kappa_\chi(\top)]\!] \to [\![\beta_\chi(\beta)\kappa_\chi(\bot)]\!]$$

$$C(\tau,\sigma^i,...,\sigma^{i+n-1}(\beta))[\![\kappa_\chi(\tau+\epsilon,\sigma)]\!] \to [\![\beta_\chi(\beta\kappa_\chi(\bot)]\!]$$

Understood that $\beta(a_1:0,a_2:x_2,...) \equiv \beta(a_2:x_2,...)$

Adjudicator * Deposit * ForceMove * Respond * Timeout * Conclude * Withdraw

2 Enforceable outcomes

We say an outcome, β_χ, is **enforceable by a participant**, p, if there exists a sequence of operations such that p can register the outcome with the adjudicator that no actions taken by another party can prevent. (Here we exclude actions that break the underlying blockchain assumptions: for example, we do not consider the possibility of censuring transactions or actions of physical violence to prevent the participant from performing those operations and so on.)

Impossible for two participants to have different enforceable outcomes.

Is possible for one participant to hold multiple enforceable outcomes.

Is possible for no outcome to be enforceable.

Is possible for one outcome to be enforceable for one participant and not others.

It's also possible for one outcome to be enforceable for all participants.

2.0.1 Safe funding

3 Turbo Protocol

Assumptions: * Can register a transaction within any blockchain interval * Transactions are free