Nitro Protocol

Tom Close

December 14, 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.

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

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

Table 1: ForceMove state format

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**.

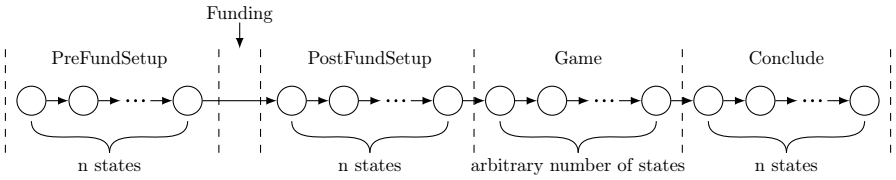

Figure 1: Overview of the stages of collaborative play in the "happy path" case. Note that the allowed transitions from `PRE/POSTFUNDSETUP` $\mapsto$ `CONCLUDE` are omitted from the diagram.

1.1 On-chain operations

Adjudicator

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

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

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

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

Unlike the other operations on this list, the timeout operation is not triggered by a blockchain transaction. Instead the operation happens automatically when the block time exceeds the expiry time stored in the challenge. In practice, there will not be any change to the state stored in the contract when the operation occurs - just a change to the interpretation of that state.

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

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

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

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

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

$\{\beta_1, \ldots \beta_n\}$ - The set of outcomes $\{\beta_1, \ldots \beta_n\}$. We say one of these outcomes is a forceable-possibility.

If

locally possible outcomes enforceable

We write this $[\beta_\chi]_p$

We say an outcome, β_χ, is **enforceable by a participant**, p, if there exists a sequence of operations [todo - not quite] 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.)

Example 2.1. Enforceability by the next mover
hello

Example 2.2. Conclusion proofs
hello

Example 2.3. PreFundSetup
hello

Example 2.4. The consensus game

$$\begin{aligned} t_{L_C}(i, \beta, (j, x), \beta', (j', x')) \Leftrightarrow [&(j = n-1 \wedge j' = 0 \wedge \beta' = x = x') \vee \\ &(j < n-1 \wedge j' = j+1, \beta = \beta', x = x') \vee \\ &(j' = 0, \beta = \beta')] \end{aligned}$$

$$\begin{aligned} \sigma^i(\beta, (0, \beta)) &\approx [\beta]_P \\ \sigma^{i+1}(\beta, (0, \beta')) &\approx \{\beta'\}_{p_0}[\beta]_{p_1, \ldots, p_{n-1}} \\ \sigma^{i+2}(\beta, (1, \beta')) &\approx \{\beta'\}_{p_0, p_1}[\beta]_{p_2, \ldots, p_{n-1}} \\ &\vdots \\ \sigma^{i+n-1}(\beta, (n-1, \beta')) &\approx [\beta', \beta]_{p_{n-1}} \\ \sigma^{i+n}(\beta', (0, \beta')) &\approx [\beta']_P \end{aligned}$$

hello

- 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.
- If you're not a participant, no channel state is enforceable

We say a state $[\![S]\!]$ is α_X-equivalent to a state $\alpha_X(x)$ if there exist a sequence of on-chain operations $O = O_i \dots O_0$ such that the α_X term in $[\![S']\!] = O[\![S]\!]$ is at least x (α_X-reachable) and there is no sequence of operations O' such that $\alpha_x(x)$ is not reachable from the resulting state $[\![S'']\!] = O'[\![S]\!]$

On-chain funding

2.1 The consensus game

3 Turbo Protocol

Turbo protocol is a small extension to ForceMove that allows multiple ForceMove state channels between a the same set of participants to be supported by a single on-chain state deposit.

* parallelized * open and close off-chain * generalisation of addresses

In Turbo the ForceMove withdrawal operation, $W_A(x)$, is replaced with two operations: a **transfer** operation, $T_{A,B}(x)$, and a modified withdrawal operation, $W'_B(x)$. The original ForceMove withdrawal can be recovered as $W_A(x) = W'_B(x)T_{A,B}(x)$.

The modified withdrawal operation, $W'_A(x)$, allows withdrawal directly from the funds held for address, A. The operation requires knowledge of the private key of A and is therefore not possible for addresses that correspond to state channels. The operation has the following effect on the on-chain state:

$$W'_A(x)[\![\alpha_A(x')]\!] \to [\![\alpha_A(x' - x)]\!]$$

The transfer operation, $T_{A,B}(x)$, is an instruction to transfer funds currently allocated to address A to address B, according to the outcome of channel A:

$$T_{AB}(x)[\![\alpha_A(x_\alpha)\beta_A(B : x_\beta, \dots)]\!] \to \begin{cases} [\![\alpha_A(x_\alpha - x)\alpha_B(x)\beta_A(B : x_\beta - x, \dots)]\!] & \text{if } x \leq x_\alpha, x_\beta \\ [\![\alpha_A(x_\alpha)\beta_A(B : x_\beta, \dots)]\!] & \text{otherwise} \end{cases}$$

Off-chain open and close

4 Nitro Protocol

Add extra type of channels Guarantor terms

`participants`	`address[]`	P	The addresses used to sign updates to the channel.
`gameLibrary`	`address`	L	The address of the gameLibrary, which defines the transition rules for this channel
`nonce`	`unit256`	k	Chosen to make the channel's address unique.
`challengeDuration`	`unit256`	η	
`turnNum`	`unit256`	i	Increments as new states are produced.
`guarantorFor`	`address`		Target channel for guarantee channels. Equal to 0 for balance channels.
`balances`	`(address, uint256)[]`	β or γ	Current *outcome* of the channel.
`isFinal`	`bool`	f	
`data`	`bytes`	δ	
`v`	`uint8`		ECDSA signature of the above arguments by the moving particpant.
`r`	`bytes32`		
`s`	`bytes32`		

Table 2: ForceMove state format

$$G_{ABC}(x)[\![\alpha_A(x_\alpha)\gamma_{A,B}(C:x_\gamma,\dots)\beta_B(\dots,C:x_\beta,\dots)]\!] \rightarrow \begin{cases} [\![\alpha_A(x_\alpha - x)\alpha_C(x)\gamma_{A,B}(C:x_\beta - x,\dots)\beta_B(\dots,C:x_\beta - x,\dots)]\!] & \text{if } x \leq x_\alpha, x_\beta, x_\gamma \\ [\![\alpha_A(x_\alpha)\gamma_{A,B}(C:x_\gamma,\dots)\beta_B(\dots,C:x_\beta,\dots)]\!] & \text{otherwise} \end{cases}$$

4.1 Virtual Channels

There is a configuration where we can support virtual channels. They can be opened and closed off-chain.

Want to fund a channel χ between A and B, for which we have state $[\beta_\chi(A:x, B:y)]_{A,B}$. We have channels L and L' with participants $\{A, C\}$ and $\{B, C\}$ respectively. We assume these channels start in states $[\beta_L(A:x, C:)]$

$$[\![\alpha_L(x)\alpha_{L'}(x)]\!] \quad [\beta_L(A:a,C:b)]_{A,C} \quad [\beta_{L'}(B:b,C:a)]_{B,C}$$

$$\begin{aligned}
&[\![\alpha_L(x)\beta_L(G:x)\gamma_{G,J}(\chi:x,C:x)\beta_J(\chi:x,C:x)\beta_\chi(A:a,B:b)]\!]\\
&\xrightarrow{T_{L,G}(x)}[\![\alpha_G(x)\gamma_{G,J}(\chi:x,C:x)\beta_J(\chi:x,C:x)\beta_\chi(A:a,B:b)]\!]\\
&\xrightarrow{G_{G,J,\chi}(a)}[\![\alpha_\chi(x)\beta_\chi(A:a,B:b)\gamma_{G,J}(C:x)\beta_J(C:x)]\!]\\
&\xrightarrow{T_{\chi,A}(a)}[\![\alpha_A(a)\alpha_\chi(b)\beta_\chi(B:b)\gamma_{G,J}(C:x)\beta_J(C:x)]\!]\\
&\approx\alpha_A(a)
\end{aligned}$$

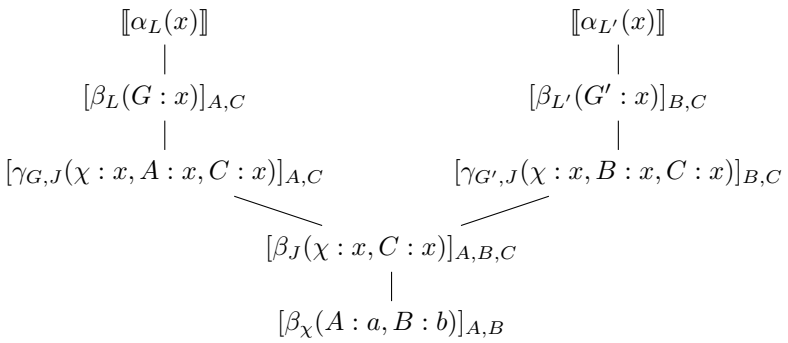

Figure 2: Ledger channels, $x = a + b$

5 Assumptions

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