Nitro Protocol

Tom Close

December 18, 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**, which we will write $\boldsymbol{\sigma}^*$.

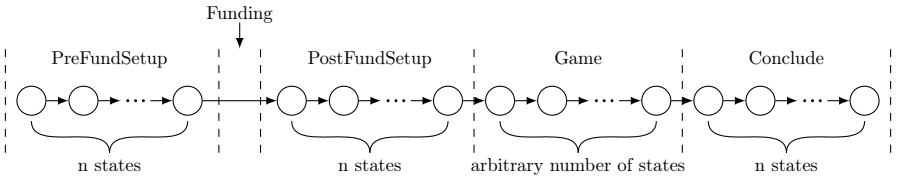

Figure 1: Overview of the stages of collaborative play in the "happy path" case. Note that the allowed transitions from `PRE/POSTFUNDSETUP` $\mapsto$ `CONCLUDE` are omitted from the diagram.

1.1 On-chain operations

todo: intro We will talk about the adjudicator as though it is a single contract but don't intend to take a position on how it should be constructed by doing so; it is likely that in practice the adjudicator functionality will be provided by a set of interacting contracts.

We will refer to the on-chain contract as the **adjudicator**, which we will write $[\![S]\!]$, where S represents the contract's internal state. The state consists of three collections: the allocations, the outcomes and the challenges, each of which is keyed by a channel address.

The **allocations** collection is an `(address -> uint256)` mapping that records the total funds allocated to a given channel. We write $[\![\alpha_\chi(x)]\!]$ to represent an adjudicator where x is allocated to channel χ.

The **outcomes** collection assigns channels to one of two different modes: channels are either undecided or decided. We use the symbol $\top$ to represent undecided channels, so that $[\![\beta_\chi(\top)]\!]$ represents an adjudicator where the outcome of channel χ is undecided. For channels with outcomes that are decided, the outcomes collection maps the channel address to an ordered set of `(address, uint256)` pairs - exactly the same format as the channel balances. We write $[\![\beta_\chi(X : x, Y : y)]\!]$ for an adjudicator where the outcome for channel χ is decided and sends x to address X and y to address Y.

The **challenges** collection records whether there is an active challenge on a given channel. We write $[\![\kappa_\chi(\tau, \sigma)]\!]$ for the case where there's an active challenge on channel χ challenging with state σ and expiring at time τ. We write $[\![\kappa_\chi(\top)]\!]$ in the case where there is no open challenge and $[\![\kappa_\chi(\bot)]\!]$ in the case where a challenge has expired (and thus no further challenges are possible). An efficient real-world implementation would most likely store the challenges as part of the outcomes collection, but we have chosen to treat them as separate concepts for the purpose of this paper.

Note that while using the above notation in the rest of the paper, we will suppress the unimportant parts of the state, with the understanding that these parts of the state remain unchanged. It is also understood that zero entries can

be ignored in outcomes, so that $\beta(a_1 : 0, a_2 : x_2, ...) \equiv \beta(a_2 : x_2, ...)$.

We now move on to describing the operations supported by the adjudicator. The first operation we look at is the **deposit**:

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

The deposit can be called by anyone and results in an increase in the allocation for the given channel.

There is also a **withdrawal** operation that can be used by anyone with the private key for address A to withraw funds allocated to A by an outcome:

$$W_A(x)[\![\alpha_\chi(x_\alpha)\beta_\chi(A : x_\beta, ...)]\!] \to \begin{cases} [\![\alpha_\chi(x_\alpha - x)\beta_\chi(A : x_\beta - x, ...)]\!] & \text{if } x \leq x_\alpha, x_\beta \\ [\![\alpha_\chi(x_\alpha)\beta_\chi(A : x_\beta, ...)]\!] & \text{otherwise} \end{cases}$$

In order to call the withdraw, the caller needs to sign the withdraw operation using the private key of the address the withdrawal is from. Thus it is not possible to call the withdraw operation for an address that corresponds to a state channel, as these addresses were generated from a hash of the channel's properties and not from a public/private key pair.

The next operation is the **force-move** which is used to register a challenge for a channel χ if no challenge (active or expired) currently exists:

$$FM(\tau, \boldsymbol{\sigma}_i^{i+n-1})[\![\kappa_\chi(x)]\!] \to \begin{cases} [\![\kappa_\chi(\tau + \eta, \sigma^{i+n-1})]\!] & \text{if } x = \top \wedge t(\boldsymbol{\sigma}_i^{i+n-1}) \\ [\![\kappa_\chi(x)]\!] & \text{otherwise} \end{cases}$$

where we write $\boldsymbol{\sigma}_i^{i+n-1}$ to represent the sequence of states $\sigma^i \dots \sigma^{i+n-1}$ and $t(\boldsymbol{\sigma}_i^{i+n-1}) = t(\sigma^i, \sigma^{i+1}) \wedge \dots \wedge t(\sigma^{i+n-2}, \sigma^{i+n-1})$.

The intent of a force-move is to get the next participant to provide their next move to the adjudicator. To do this they can call the **respond** method, providing their next state:

$$R(\tau', \sigma^{i+1})[\![\kappa(\tau, \sigma^i)]\!] \to \begin{cases} [\![\kappa_\chi(\top)]\!] & \text{if } \tau' < \tau \wedge t(\sigma^i, \sigma^{i+1}) \\ [\![\kappa(\tau, \sigma^i)]\!] & \text{otherwise} \end{cases}$$

Once the challenge is cancelled it is removed from the adjudicator and the channel operation can continue off-chain. The force-move paper [] outlines a couple of alternative ways (**refute** and **respond-with-alternative-move**) to respond to a force-move. While these are important for the ForceMove protocol, we will not need them in this paper, so will not go over them here.

If the opponent does not respond to the challenge before the challenge expiry time, τ, then a **timeout** occurs:

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

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.

The final operation is the **conclude** operation, which enables the immediate creation of a channel outcome from a conclusion proof, $\boldsymbol{\sigma}^*(\beta)$ including in the case where there is an active challenge:

$$C(\tau, \boldsymbol{\sigma}^*(\beta))[\![\kappa_\chi(x)]\!] \to \begin{cases} [\![\kappa_\chi(\bot)]\!] & \text{if } x = \bot \\ [\![\beta_\chi(\beta)\kappa_\chi(\bot)]\!] & \text{otherwise} \end{cases}$$

The conclude operation enables instant withdrawals in the case the channel is concluded collaboratively off-chain.

2 Enabled and Enforceable Outcomes

Analyse what a state means at a given point in time. Involves reasoning about the states and private information they hold and their knowledge about what other players hold. Don't track in detail - from a player's point of view it's us vs world.

Call an outcome of a channel the balances that end up in the adjudicator. Either via a challenge + expiry or by the conclude operation.

We introduce these concepts using the ForceMove framework but they are the central concepts of all state channel logic and can be applied to all state channel systems.

2.1 Enforceable Outcomes

Enforceability of an outcome, β is **enforceable** by p: * it is possible for p to finalize β in the adjudicator $[\![\beta_\chi(\beta)]\!]$ and no-one else can stop them. We write this $[\beta]_p$.

Example: p is the next player to move

It is possible for an outcome to be enforceable for multiple players at the same time:

Example: a single conclusion proof exists

Example: after the post fund setup

Note that it is also possible for more than one outcome to be enforceable by a player. We will see an example of this later. It follows from the definition of

enforceability that it is impossible to simultaneously have one player with more than one enforceable outcome and more than one player at least one enforceable outcomes.

It's also possible to get into a state where no outcomes are enforceable by anyone. This is usual undesirable and a sign that something somewhere has gone wrong. An example of this is the case where two different conclusion proofs exist.

2.2 Enabled Outcomes

It is often the case that a given player, p, will have no enforceable outcomes at a given point in time. In this case we instead look at the set of **enabled** outcomes for p, which we write $\{\beta_1, ...\beta_i\}_p$. The enabled outcomes are the set of all outcomes of the channel that are possible up until p signs their next state. They are the outcomes that were enabled by p's last signature.

All the possible ways the channel can end before they take their next turn - that they can't control(?).

The set of enabled outcomes has the property that p can cause one of them to be registered with the adjudicator (e.g. through repeated force-moving) but cannot choose which one.

Sometimes a player might have incomplete knowledge, so they are not able to calculate the possible states. In this case the enabled outcomes should include all possible states as far as the player can tell. An example of this occurs in commit-reveal schemes: say two players are playing a guessing game where player A commits to a value, player B makes a guess and then player A reveals whether they were right. At the point where B has just made their guess, the outcome is determined but B does not yet know what it is. In this case, the enabled outcomes for B should include the outcome where B was correct and the outcome where B was incorrect.

If a player has enforceable outcomes, they do not have any enabled outcomes. If a player has only one enabled outcome then, by the definitions, it is an enforceable outcome: $\{\beta\}_p \equiv [\beta]_p$.

2.3 The consensus game

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

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

2.4 W-Equivalence

We say a state $[\![S]\!]$ is α_X-equivalent to a state $\alpha_X(x)$ if there exist a sequence of on-chain operations $O = O_i \ldots O_0$ 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 $[\![S'']\!] = O'[\![S]\!]$

How much could I withdraw on-chain right now.

Example: $[\![\alpha_\chi(x)]\!][\beta_\chi(A:x)]_A \sim_A x$

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:

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

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 participant.
`r`	`bytes32`		
`s`	`bytes32`		

Table 2: ForceMove state format

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

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 $[\beta_L(A : x, C :)]$

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

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

5 Assumptions

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

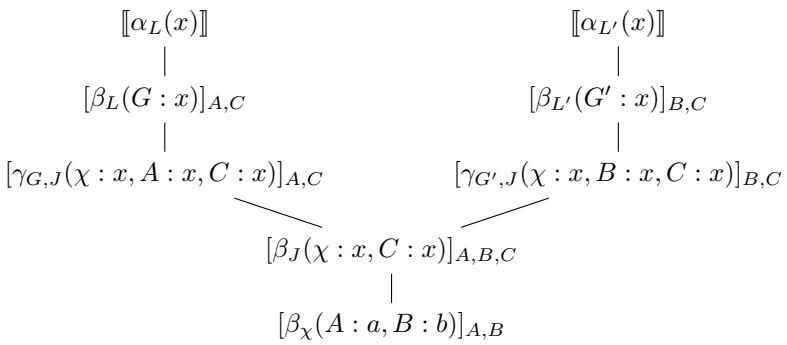

Figure 2: Ledger channels, $x = a + b$