The following is an excerpt from the upcoming paper on Nitro Protocol, which describes how to build state channel networks.

3 Modelling State Channel Networks

In this section will present a simple model to form the foundation of a state channel network. The model is intended to be easy to understand and reason about, while still capturing the essential features. It will form the basis for the protocols introduced later in the paper, as well as for the tools used to prove the correctness of those protocols.

3.1 A System of Balances

At the heart of our model lies a simple system of balances. We start by describing that system.

In this paper, we simplify the explanation by only considering a single asset, which we will refer to as **coins**. We will further simplify matters by specifying that quantities of coins will have no maximum size, taking values in $\mathbb{Z}^+$. This allows us to avoid dealing with integer overflows when presenting operations. These simplifications do not cause any limitations in practice and all the work here can be applied to state channel networks that manage an arbitrary number of asset types.

In order to store value, a state channel network must be backed by assets held on-chain. In our explanation, we assume that these funds are held and managed by a single[1] smart contract, which we will refer to as the **adjudicator**.

The first purpose of the adjudicator is to store the balance of coins held for a given address. Addresses can correspond either to participants in the network or to state channels. A **participant address** is a regular blockchain address, generated in the standard way from the signature scheme. A **channel address** is formed by taking the hash of the participant addresses along with a nonce, k, that is chosen by the participants in order to distinguish their channels from one another. Given an address, A, we assume that the properties of the signature scheme and hashing algorithm make it impossible to find a private key for A or to construct a channel whose address is A, if they are not previously known.

We model the adjudicator as having a simple mapping that stores the quantity of coins for an address. If an address does not appear in the table we take the balance to be zero. Figure 1 introduces the notation we will use to describe the system.

[1] We assume this for simplicity only - in practice the functionality could be split across multiple contracts.

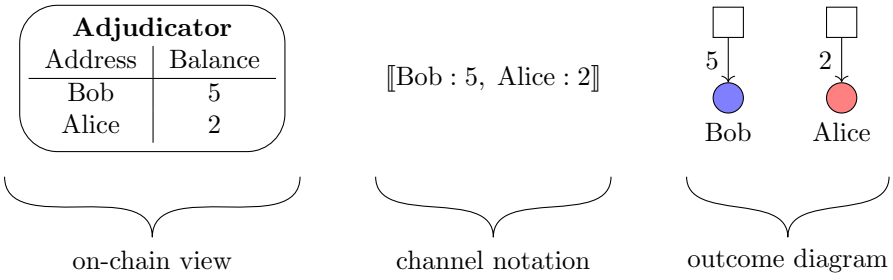


Figure 1: *Three different ways of representing the situation where Bob has* 5 *coins stored against his address in the adjudicator and Alice has* 2*. The on-chain view shows a pictorial representation of the state in the adjudicator. Channel notation is useful for writing the state in equations and will later be extended to cover off-chain state as well. In the outcome diagram, the white squares represent adjudicator balances and the solid circles are coloured to represent the different participants.*

The **deposit** operation, $D_A(x)$, is an on-chain operation used to increase A's balance by x coins. There are no restrictions on who can deposit coins for an address, but the transaction must always include a transfer of x coins into the adjudicator.

The **withdrawal** operation, $W_A(x)$, can be used to withdraw coins held at participant address, A, by any party with the knowledge of the corresponding private key. In practice the withdrawal should also specify the blockchain address where the funds should be sent. A potential method signature is `withdraw(fromAddr, toAddr, amount, signature)`, where `signature` is A's signature of the other parameters [2].

To recap, we now have a simple smart contract that can store a balance against an address, which either represents a participant or a channel. The balances can be increased and decreased through deposits and withdrawals. Anyone can deposit into an address of either type[3] but withdrawals can only occur from participant addresses - and only by a party who knows the private key. The total coins held by the smart contract should always equal the sum of the balances.

3.2 State Channel Outcomes

In our model, a state channel is an off-chain protocol followed by a set of participants, enabling them to reach an **outcome** that can be used to update the

[2] In practice, we could add the `senderAddress` to the parameters to sign, in order to prevent replay attacks by other parties.

[3] But there is nothing to be gained from depositing into a participant address.

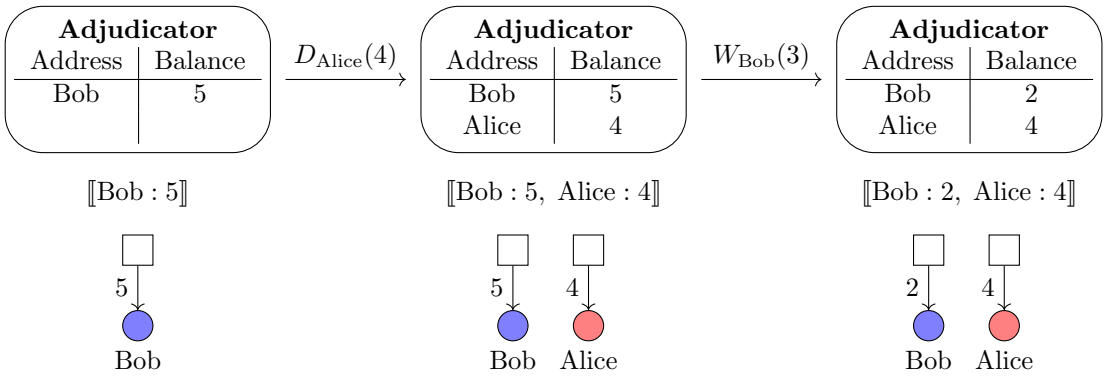
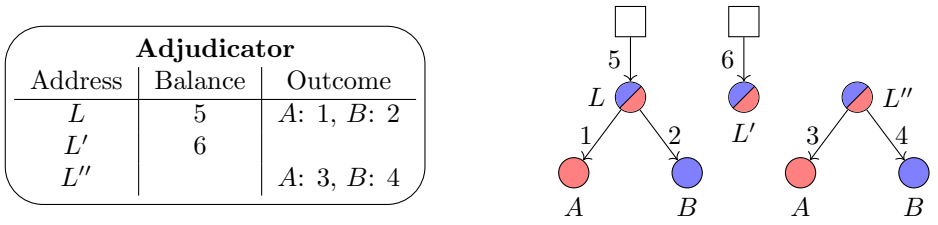


Figure 2: *Deposits and withdrawals. The deposit can be called by any blockchain user, provided that it is accompanied with a transfer of the same number of coins into the adjudicator. For the withdrawal to be successful, the withdrawal parameters must be signed with Bob's private key.*

balances on the chain. The format and interpretation of the outcome is specified by the state channel network protocol being used. An example of a type of outcome is the *allocation*, which is used in both Turbo and Nitro protocol, and which consists of a list of recipient addresses and totals that specify how the channel's balance should be distributed.



$$\llbracket \text{L} : 5 \mapsto (A : 1, B : 2),\ L' : 6,\ L'' \mapsto (A : 3, B : 4) \rrbracket$$

Figure 3: *Representation of (allocation) outcomes in the three different diagram formats. We show the three possible cases: a channel, L, with both a balance and an outcome; a channel, L', with a balance but no outcome; and a channel, L'', with an outcome but no balance. We represent the channels with split circles, coloured to represent the participants of the channel, which we have taken to be A and B.*

Central to the approach of the model is to split the updating of the balances into two steps:

1. **Finalization**: getting to a point where the outcome of the channel is

stored on the blockchain.

2. **Redistribution**: updating the balances in the adjudicator according to that outcome.

The bulk of this paper is focussed on the redistribution step, which we will discuss further in section 3.3.

Understanding how an outcome is finalized inevitably involves understanding the rules of operation of the state channel: from exchanging states to launching and responding to challenges. The ForceMove protocol completely specifies these rules of operation, in a way that is compatible with the work in this paper.

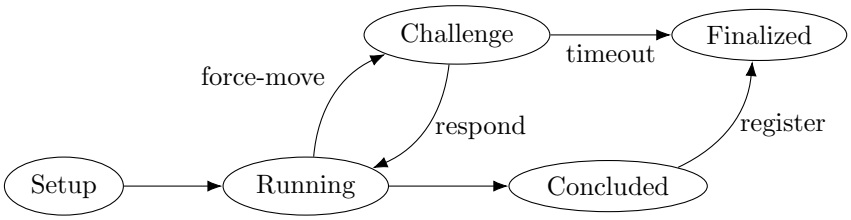


Figure 4: *ForceMove Channel Operation. There are two ways for an outcome to become finalized: (i) through a challenge that times out before anyone responds, and (ii) through the registration of a conclusion proof.*

In ForceMove, there are two ways for an outcome to be finalized. The first corresponds to a non-collaborative closing of the channel: one participant launches an on-chain challenge by providing a sequence of signed statements to the force-move operation; this starts a timeout period; if no other participant responds during timeout period, then the challenge times out and the outcome corresponding to the challenge state is finalized. The second corresponds to a collaborative closing of the channel: all the participants sign a special conclude state, which contains the channel outcome to produce an object called a conclusion proof; any participant can then register this outcome on-chain to create a finalized outcome. By closing the channel collaboratively the participants avoid having to wait for the challenge period.

For the purpose of the model, it is crucial that the rules of operation only allow one outcome to be finalized for each channel. As we will see in section **??**, it is also important that the rules of operation make it possible to know which outcome(s) a participant can finalize from a given state.

3.3 Redistribution

The second part of extracting the funds from a state channel is the redistribution step. Redistribution involves calling a sequence of on-chain **operations** to

manipulate the balances and finalized outcomes. The allowed operations are defined by the network protocol used. Figure 5 shows an example of the transfer operation from Turbo protocol.

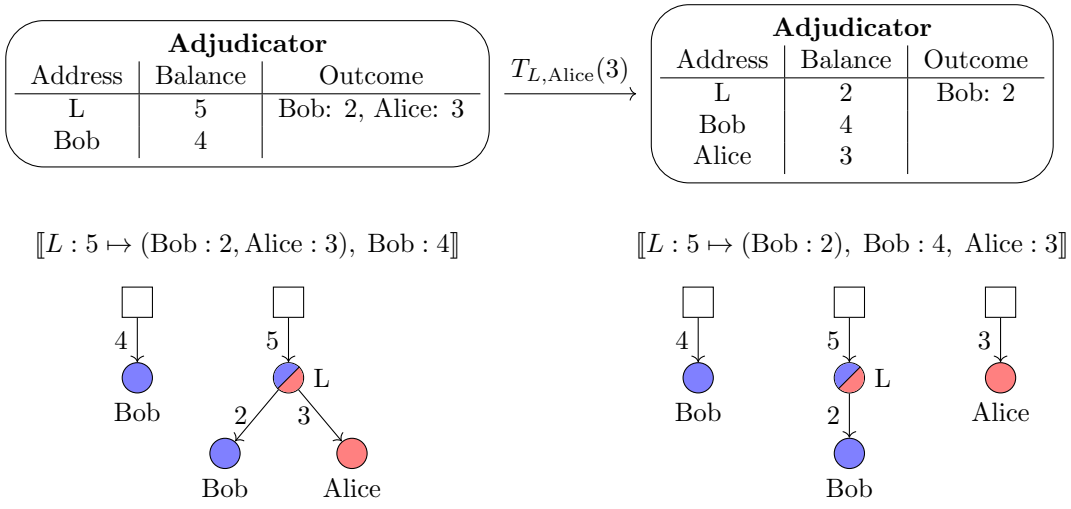


Figure 5: *A example of a redistribution operation from Turbo protocol. Here the transfer operation is used to move* 3 *coins out of channel* L *to Alice. Note: the parts of the adjudicator responsible for storing challenges are omitted from the diagram.*

The operations change the state of the adjudicator: typically both balances and outcomes. There is no restriction on who can trigger the operations. In this paper, we present all operations separately and assume they are called separately. In practice, we expect that implementations would provide some utility methods that combine common sequences of operations, to improve gas efficiency.

To recap, we now have a system where participants can deposit into state channel addresses on-chain. By running a state channel, participants can reach an outcome and ensure this outcome is finalized on-chain. Once the outcome is finalized, the funds held in the channel can be redistributed to other participants and channels by calling on-chain operations. Participants can then withdraw any funds that have been redistributed to their address.

In the next section, we will use this model to develop some tools for constructing state channel networks and proving their safety. In particular, we will develop the logic by which we can use the fact that a given outcome could be finalized if necessary, to avoid putting that outcome on-chain at all.