6 Nitro Protocol

Nitro protocol is an extension to Turbo protocol. In Nitro protocol, the outcome of a channel can be either an allocation or a **guarantee**. There are two on-chain redistribution operations: the transfer and the **claim**.

Nitro enables true state channel networks with virtual channels, where channels are routed through intermediaries. In particular, the extra features of Nitro allow virtual channels to be safely opened and closed off-chain while maintaining the property that channels update independently[1].

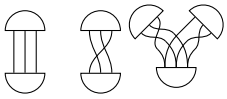
6.1 Guarantees and Claims

- guarantees are paired with allocations - for a guarantee outcome to pay out funds, it needs to ba paired with a allocation outcome - target a given outcome - allows them to pay out in a different priority

- complicated in that we may not know the precise outcome or even the addresses that will appear in it at the time we create the guarantee - because of this we need to record the guarantee to apply a range of outcomes - use the format of a list of addresses, that will be moved to top priority

- if guarantee specifies [a, b, c] and the outcome is (c: 1, d:2, a:4), then the guarantee will pay out as though the outcome was (a:4, c:1, d: 2)

- can afford



Nitro adds the **claim** operation, $C_{G,A}(x)$, to the existing transfer, deposit and withdraw operations. If G acts as guarantor for L and can afford x for A, then $C_{G,A}(x)$ has the following three effects:

- Reduces the funds held in channel G by x.
- Increases the funds held in channel A by x.
- Reduces the amount owed to A in the outcome of L by x.

Otherwise, the claim operation has no effect.

6.2 Redistributing

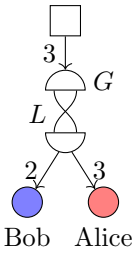
Reasoning about redistribution in Nitro is more complicated than in Turbo. For a start, it is possible to construct situations where the same outcome can lead

[1] It is possible to implement virtual channels in Turbo but only by breaking the independent update constraint. We believe this constraint will be important when running large scale state channel networks.

Adjudicator

Address	Balance	Outcome
G	3	$(L\|\text{Alice})$
L		Bob: 2, Alice: 3

$[\![G : 3 \mapsto (L|\text{Alice}),\ L \mapsto (\text{Bob} : 2, \text{Alice} : 3)]\!]$



Adjudicator

Address	Balance	Outcome
G		$(L\|\text{Alice})$
L		Bob: 2
Alice	3	

$[\![G \mapsto (L|\text{Alice}),\ L \mapsto (\text{Bob} : 2),\ \text{Alice} : 3]\!]$

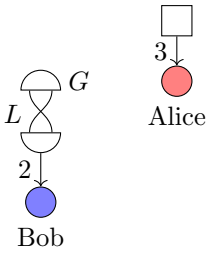


Figure 1: *The claim operation*

to different values, depending on the order in which guarantees are claimed. Figure 2 shows one of these situations.

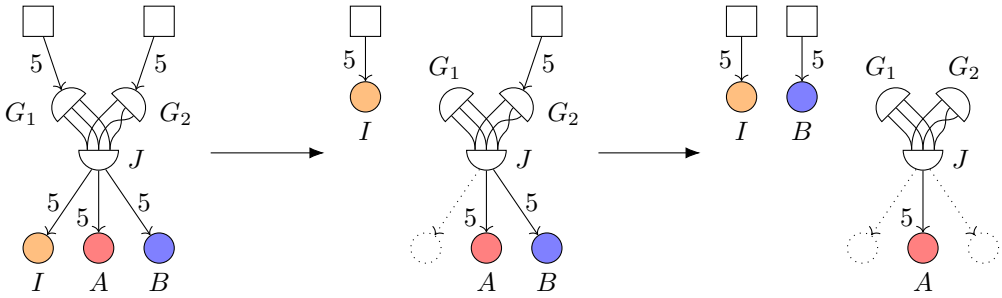


Figure 2: *Guarantee claim ordering problem. In the diagram both G_1 and G_2 guarantee J's outcome with I as first priority but with different second priorities. If G_1 is claimed first, then when G_2 is claimed the funds go to B. If G_2 is claimed first, then when G_1 is claimed the funds go to A. Whether A or B ultimately gets paid depends on the order that the guarantees are claimed.*

Despite this issue, it is still possible to make some statements about redistribution in Nitro, in particular putting some lower bounds on how funds are distributed. For example, in the example in figure 2 one thing we can definitely say is that participant I will receive their 5 coins, even though we cannot say anything about how the remaining 5 coins will be distributed between A and B. These lower bounds prove to be enough to handle the constructions used in

the rest of the chapter.

We can calculate this lower bound with a modification the Turbo Value Algorithm. The key idea is when handling outcomes that are targets of one or more guarantees, to perform a separate analysis

Minimum Payout Calculation

In this calculation we will consider an allocation channel, L, whose outcome allocates $a_1 \dots a_m$ to destination addresses $D_1 \dots D_m$, and a set of guarantees $G_1 \dots G_n$ which target L. We want to calculate the minimum payout, p_i, that each destination will receive when all possible payout orders are considered.

Each guarantee, G_i, induces a permutation π_i on the destination addresses, so that G_i prioritizes the outcomes in the order $D_{\pi_i(1)} \dots D_{\pi_i(m)}$.

We start in a state where the values of channel L and the guarantees are known, with G_i having value $v_i = \texttt{Value}[G_i]$. We will assume the value of L itself is 0. We are free to do this because, if $\texttt{Value}[L] = x > 0$, then for the purpose of running the algorithm we can write the problem in an equivalent way, by adding a guarantee G_{n+1} that has value x, that targets L and that has $\pi_{n+1}(k) = k$.

If $\sum v_i > \sum a_j$, then we say the system is overfunded. In this case, we know that all destinations will receive their allocations, so $p_i = a_i$ regardless of the order of payout. Otherwise, we let $p_{ij} > 0$ be the amount paid out from guarantee G_i to destination D_j and introduce the following set of constraints to ensure that we only consider situations where no funds are left in the guarantees:

$$\sum_j p_{ij} = v_i \tag{1}$$

It is useful to introduce the deficit, $\delta_j > 0$, for the destination D_j, defined by the equations:

$$\delta_j + \sum_i p_{ij} = a_j \tag{2}$$

Finally we can write down the a set of constraints that encode the priority order of the guarantees:

$$\begin{aligned} p_{i\pi_j(2)} > 0 &\Rightarrow \delta_{\pi_j(1)} = 0 \\ p_{i\pi_j(3)} > 0 &\Rightarrow \delta_{\pi_j(2)} = \delta_{\pi_j(1)} = 0 \\ &\vdots \\ p_{i\pi_j(m)} > 0 &\Rightarrow \delta_{\pi_j(m-1)} = \dots = \delta_{\pi_j(1)} = 0 \end{aligned} \tag{3}$$

We can then calculate $p_i = a_i - \delta_i^*$, where δ_i^* is found by minimising δ_i subject to these constraints.

In general, calculating the minimimum payout therefore involves solving a constrained optimization problem, with non-linear constraints. In practice, for all the calculations required for the constructions in this paper, it is sufficient to look at two special cases: (i) when the allocation is fully funded and (ii) when there is only a single guarantee. In the fully funded case, where $\sum v_i = \sum a_i$, it is easy to see that $p_i = a_i$, just as in the overfunded case. In the single guarantee case, the payouts are fully determined, so it is easy to calculate the minimum payout.

- in general this is a non-linear optimization - suffice to look at two special cases: fully funded and single guarantee

Example

$$[\![G_1 : 5 \mapsto (J|I, A, B),\ G_2 : 5 \mapsto (J|I, B, A),\ J \mapsto (I : 5, A : 5, B : 5)]\!] \tag{4}$$

Let $D_1 = I$, $D_2 = A$, $D_3 = B$.

$$\delta_1 + p_{11} + p_{21} = 5 \tag{5}$$
$$\delta_2 + p_{12} + p_{22} = 5 \tag{6}$$
$$\delta_3 + p_{13} + p_{23} = 5 \tag{7}$$

$$p_{11} + p_{12} + p_{13} = 5 \tag{8}$$
$$p_{21} + p_{22} + p_{23} = 5 \tag{9}$$

$$p_{12} > 0 \Rightarrow \delta_1 = 0 \tag{10}$$
$$p_{13} > 0 \Rightarrow \delta_2 = \delta_1 = 0 \tag{11}$$
$$p_{23} > 0 \Rightarrow \delta_1 = 0 \tag{12}$$
$$p_{22} > 0 \Rightarrow \delta_3 = \delta_1 = 0 \tag{13}$$

First suppose that $\delta_1 > 0$. Then equations (10) to (13) give that $p_{12} = p_{13} = p_{23} = p_{22} = 0$. Substituting into equation (8) gives that $p_{11} = 5$ and, similarly, equation (9) gives that $p_{21} = 5$. But then equation (5) cannot possibly hold. So $\delta_1 = 0$ in all cases and therefore $\delta_1^* = 0$.

- key insight is to form groups comprising of a ledger channel plus all the guarantees that target the ledger channel - we then use a different rule to

- suffices to get lower bound - follow the same algorithm as before but with guarantees treated as units - calculate the minimum that comes out to each downstream address

6.3 Virtual Channels

A virtual channel is a channel between two participants who do not have a shared on-chain deposit, supported through an intermediary. We will now give the construction for the simplest possible virtual channel, between A and B through a shared intermediary, I. Our starting point for this channel is a pair of ledger channels, L and L', with participants $\{A, I\}$ and $\{B, I\}$ respectively.

$$[\![L : x, L' : x]\!],\ [L \mapsto (A : a, I : b)]_{A,I},\ [L' \mapsto (B : b, I : a)]_{B,I} \tag{14}$$

where $x = a + b$. The participants want to use the existing deposits and ledger channels to fund a virtual channel, χ, with x coins.

In order to do this the participants will need three additional channels: a joint allocation channel, J, with participants $\{A, B, I\}$ and two guarantor channels G and G' which target J. The setup is shown in figure 3.

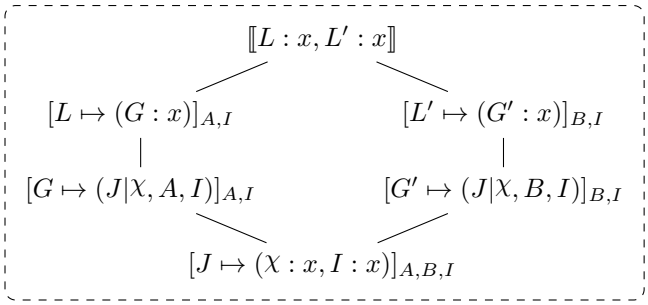


Figure 3: *Virtual channel construction*

We will cover the steps for safely setting up this construction in section 6.5. In the next section, we will explain why this construction can be considered to fund the channel χ.

6.4 Offloading Virtual Channels

Similarly to the method for ledger channel construction, we will show that the virtual channel construction funds χ by demonstrating how any one of the participants can offload the channel χ, thereby converting it to an on-chain channel that holds its own funds.

We will first consider the case where A wishes to offload χ. A proceeds as follows:

1. A starts by finalizing all their finalizable outcomes on-chain:

$$[\![L : x \mapsto (G : x),\ L' : x, G \mapsto (J|\chi, A, I),\ J \mapsto (\chi : x, I : x)]\!] \tag{15}$$

   Although A has the power to finalize L, G and J, they are not able to finalize L'. Thankfully, this does not prevent them from offloading χ.

2. A then calls $T_{L,G}(x)$ to move the funds from L to G:

$$[\![L' : x, G : x \mapsto (J|\chi, A, I),\ J \mapsto (\chi : x, I : x)]\!] \tag{16}$$

3. Finally A calls $C_{G,A}(\chi)$ to move the funds from G to χ.

$$[\![L' : x, G \mapsto (J|\chi, A, I),\ J \mapsto (I : x),\ \chi : x]\!] \tag{17}$$

As G has χ as top priority, the operation is successful.

By symmetry, the previous case also covers the case where B wants to offload. The final case to consider is the one where I wants to offload the channel and reclaim their funds. This is important to ensure that A and B cannot lock I's funds indefinitely in the channel.

1. I starts by finalizing all their finalizable outcomes on-chain:

$$[\![L : x \mapsto (G : x),\ L' : x \mapsto (G' : x), G \mapsto (J|\chi, A, I),\\ G' \mapsto (J|\chi, B, I),\ J \mapsto (\chi : x, I : x)]\!] \tag{18}$$

2. I then transfers funds from the ledger channels to the virtual channels by calling $T_{L,G}(x)$ and $T_{L',G'}(x)$:

$$[\![G : x \mapsto (J|\chi, A, I),\ G' : x \mapsto (J|\chi, B, I),\ J \mapsto (\chi : x, I : x)]\!] \tag{19}$$

3. Then I claims on one of the guarantees, e.g. $C_{G,\chi}(x)$ to offload χ:

$$[\![G \mapsto (J|\chi, A, I),\ G' : x \mapsto (J|\chi, B, I),\ J \mapsto (I : x),\ \chi : x]\!] \tag{20}$$

4. After which, I can recover their funds by claiming on the other guarantee, $C_{G',I}(x)$:

$$[\![G \mapsto (J|\chi, A, I),\ G' \mapsto (J|\chi, B, I),\ \chi : x,\ I : x]\!] \tag{21}$$

Note that I has to claim on both guarantees, offloading χ before being able to reclaim their funds. The virtual channel became a direct channel and the intermediary was able to recover their collateral.

6.5 Opening and Closing Virtual Channels

In this section we present a sequence of network states written in terms of universally finalizable outcomes, where each state differs from the previous state only in one channel. We claim that this sequence of states can be used to derive a safe procedure for opening a virtual channel, where the value of the network remains unchanged throughout for all participants involved. We justify this claim in the appendix.

The procedure for opening a virtual channel is as follows:

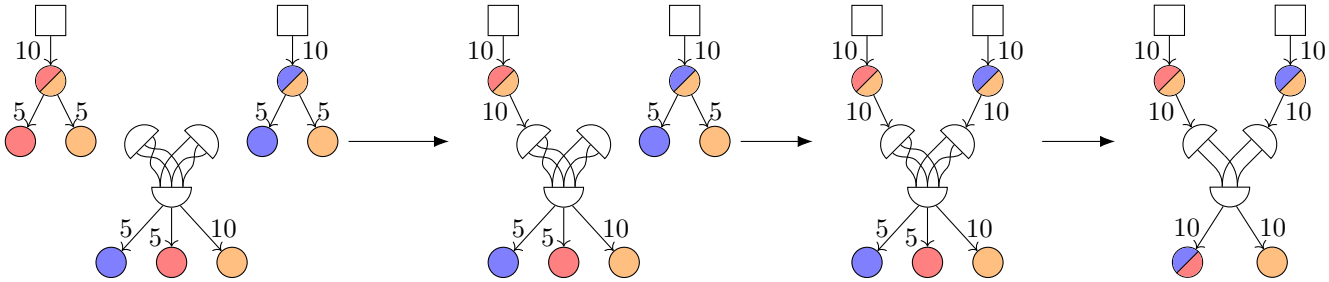


Figure 4: *Opening a virtual channel*

1. Start in the state given in equation (14):

$$[\![L : x, L' : x]\!] \tag{22}$$
$$[L \mapsto (A : a, I : b)]_{A,I} \tag{23}$$
$$[L' \mapsto (B : b, I : a)]_{B,I} \tag{24}$$

2. A and B bring their channel χ to the funding point:

$$[\chi \mapsto (A : a, B : b)]_{A,B} \tag{25}$$

3. In any order, A, B and I setup the virtual channel construction:

$$[J \mapsto (A : a, B : b, I : x)]_{A,B,I} \tag{26}$$
$$[G \mapsto (J|\chi, A, I)]_{A,I} \tag{27}$$
$$[G' \mapsto (J|\chi, B, I)]_{B,I} \tag{28}$$

4. In either order switch the ledger channels over to fund the guarantees:

$$[L \mapsto (G : x)]_{A,I} \tag{29}$$
$$[L' \mapsto (G' : x)]_{B,I} \tag{30}$$

5. Switch J over to fund χ:

$$[J \mapsto (\chi : x, I : x)]_{A,B,I} \tag{31}$$

We give a visual representation of this procedure in figure 5.

The same sequence of states, when taken in reverse, can be used to close a virtual channel:

1. Participants A and B finalize χ by signing a conclusion proof:

$$[\chi \mapsto (A : a', B : b')]_{A,B} \tag{32}$$

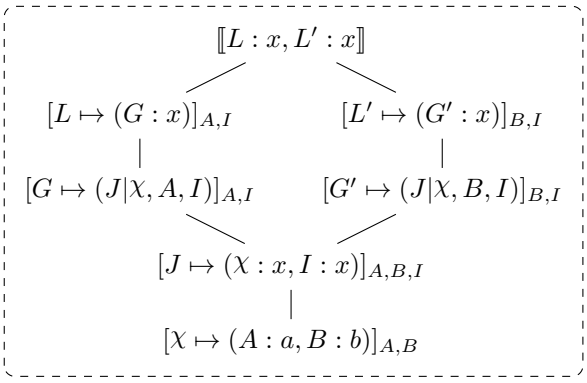


Figure 5: *Opening a virtual channel*

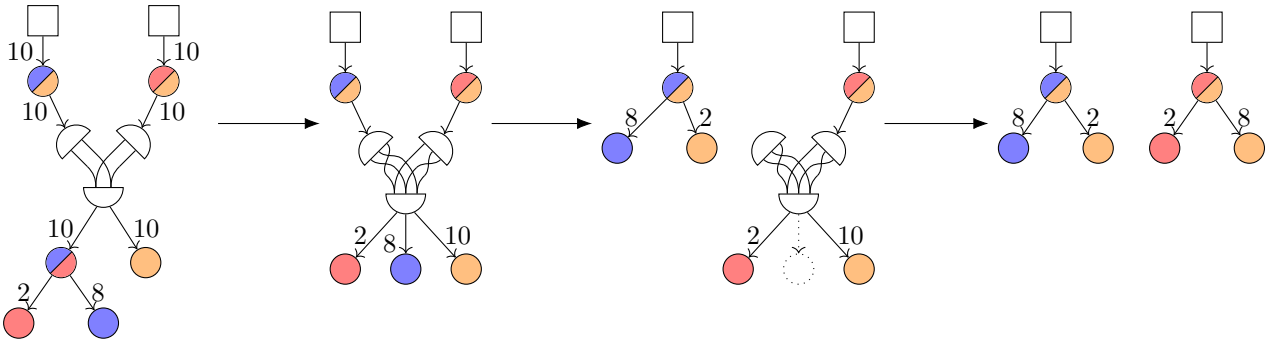


Figure 6: *Closing a virtual channel*

2. A and B sign an update to J to take account of the outcome of χ. I will accept this update, provided that their allocation of x coins remains the same:

$$[J \mapsto (A : a', B : b', I : x)]_{A,B,I} \tag{33}$$

3. In either order switch the ledger channels to absorb the outcome of J, defunding the guarantor channels in the process:

$$[L \mapsto (A : a', I : b')]_{A,I} \tag{34}$$
$$[L' \mapsto (B : b', I : a')]_{B,I} \tag{35}$$

4. The channels χ, J, G and G' are now all defunded, so can be discarded

It is also possible to do top-ups and partial checkouts from a virtual channel.

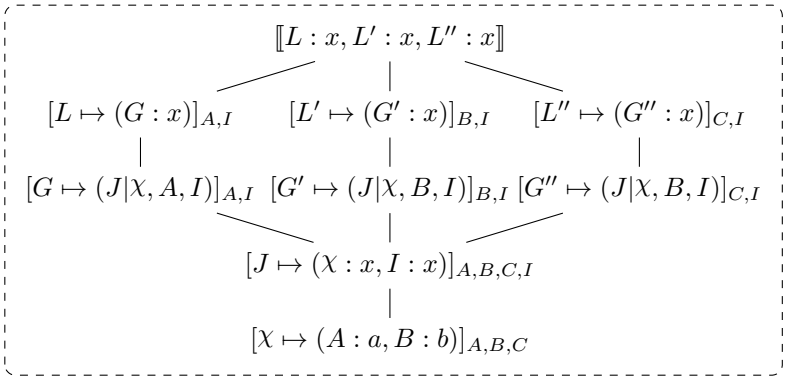


Figure 7: *Virtual channel with three participants*