- motivation - what are virtual channels vs ledger channels vs direct channels

- direct channels - channel holds money on-chain - direct help in certain circumstances but aren't a general

- ledger channels - multiple channels between a given set of participants to be supported by a single on-chain deposit - channels update independently

- virtual channels -

- number of deposits held on-chain

  - **Direct channels**: one on-chain deposit per channel
  - **Ledger channels (Turbo)**: one on-chain deposit per group of participants
  - **Virtual channels (Nitro)**; one on-chain deposit per participant

- assume network of hubs

1 Funding vs Operation

At the heart of our approach lies the decision to make the funding of a state channel independent from its operation.

We view a state channel as a device for allowing a fixed set of participants to determine how a set of shared assets should be split between them. We refer to the set of instructions that determine how the funds should be split as the **outcome** the state channel. We define the **operation** of a state channel to be the process by which the channel reaches an outcome. The **funding** of a state channel is the method of ensuring that the outcome is honoured on-chain.

Specifying the operation of a state channel involves specifying the format of the states and the update rules that define the allowed transitions between these states. Specifying the operation also involves defining the rules surrounding on-chain challenges and the ways to respond to them. The ForceMove protocol is an example of a protocol that specifies the operation of a state channel. In this paper, we will not specify the operation of state channels, instead leaning on ForceMove where required.

Specifying how state channels are funded involves specifying how funds are held in escrow on the chain, how they can be deposited and how they can be claimed according to the outcome of a state channel. As we will see in this paper, it can also involve specifying the rules for how the outcomes of multiple channels interact, which enables channels to be funded and defunded without on-chain operations. In the ForceMove paper, all funding was done by the *SimpleAdjudicator*, which only allowed for direct channels between participants.

Decoupling the funding of the channel from its operation has several advantages. Provided that a channel is funded, its operation is completely independent from the other channels in the system. The source of funding for a channel can even change from off-chain to on-chain, all without interrupting the operation of that channel. Overall, by forcing channels to operate independently and only be coupled through funding relationships, it becomes a lot easier to reason about the behaviour of any applications running within a state channel.

2 Turbo

The outcome of a Turbo channel is always an **allocation**. An allocation is a list of pairs of addresses and totals, $(a_1{:}v_1, \ldots, a_m{:}v_m)$, where the total, v_i, represents that amount of coins due to the address, a_i. If the outcome of channel A includes the pair $B{:}x$, we say that 'A **owes** x to B'. We assume that each address only appears once in the allocation and require that implementations enforce this by ignoring any entries for a given address after the first.

The allocation is in priority order, so that if the channel does not hold enough funds to pay all the coins that are due, then the addresses at the beginning of the allocation will receive funds first. We say that 'A **can afford** x for B', if B would receive at least x coins, were the coins currently held by A to be paid out in priority order.

Turbo introduces the **transfer** operation, $T_{A,B}(x)$, to trigger the on-chain transfer of funds according to an allocation. If A can afford x for B, then $T_{A,B}(x)$: (a) reduces the funds held by A by x, (b) increases the funds held by B by x, and (c) reduces the amount owed to B in the outcome of A by x. If A cannot afford x for B, then $T_{A,B}(x)$ fails, leaving the on-chain state unchanged.

Example 2.1. In the following example, we have a channel, L, which holds 10 coins and has an outcome, $(A : 3, B : 2, \chi : 5)$, which has been registered on-chain. As L can afford 4 for χ the following transfer operation is successful:

$$T_{L,\chi}(4)[\![L{:}10 \mapsto (A : 3, B : 2, \chi : 5)]\!] = [\![L{:}6 \mapsto (A : 3, B : 2, \chi : 1), \chi{:}4]\!]$$

We give a python implementation of the Turbo adjudicator in the appendix.

2.1 Ledger Channels

A **ledger** channel is a channel who uses its own funding to fund other channels sharing the same set of participants. By doing this, a ledger channel allows these **sub-channels** to be opened, funded and closed without any on-chain operations.

To see how this works, consider the following setup where a ledger channel, L, allocates the funds it holds on-chain to participants A and B and channel χ:

$$[\![L{:}10]\!], [L \mapsto (A:2, B:3, \chi:5)]_{A,B}$$

We have chosen the simplest example here, where L is funded by the coins it holds on-chain, but it is completely possible to have ledger channels themselves funded by other ledger channels or (later) by virtual channels.

We claim that the ledger channel L funds the channel χ in the above setup. This is because either participant has the power to convert the above set of states into a situation where χ is funded on-chain:

$$T_{L,\chi}(5)[\![L{:}10 \mapsto (A:2, B:3, \chi:5)]\!] = [\![L{:}10 \mapsto (A:2, B:3), \chi{:}5]\!]$$

After this operation, we say that the channel χ has been **offloaded**. Note that we do not need to wait for χ to complete before offloading it. The offload converts χ from an off-chain sub-channel to an on-chain direct channel, without interrupting its operation in any way.

The offload should be seen as an action of last-resort. It is important that offloading is allowed so that either player can realize the value in channel χ if required, but it has the downside of forcing all sub-channels supported by L to be closed on-chain. It is in the interest of both participants to open and close sub-channels collaboratively. We next show how this can be accomplished safely.

In Turbo, all ledger channels are regular ForceMove channels running the Consensus Application.

2.1.1 Opening a sub-channel

The value of a ledger channel comes from the ability to open and close sub-channels without on-chain operations. Here we show how to open a sub-channel.

1. Start in a state where A and B have a funded ledger channel, L, open:
   - $[\![L{:}x]\!],\ [L \mapsto (A:a, B:b)]_{A,B}$
2. A and B create their sub-channel χ and progress it to the funding point. We assume that $a' \leq a$ and $b' \leq b$:
   - Create channel χ: $[\chi \mapsto (A:a', B:b')]_{A,B}$
3. Update the ledger channel to fund the sub-channel:
   - Update L: $[L \mapsto (A:a-a', B:b-b', \chi:a'+b')]_{A,B}$

2.1.2 Closing a sub-channel

When the interaction in a sub-channel, χ, has finished we need a safe way to update the ledger channels to incorporate the outcome. This allows the sub-channel to be defunded and closed off-chain.

1. We start in the state where χ is funded via the ledger channel, L:
   - $[\![L{:}x]\!]$, $[L \mapsto (A : a, B : b, \chi : c)]_{A,B}$, where $x = a + b + c$.
2. The next step is for A and B to concluded channel χ, leaving the channel in the concluded state.
   - Update channel χ: $[\chi \mapsto (A : a', B : b')]_{A,B}$, where $a' + b' = c$.
3. The participants then update the ledger channel to include the result of channel χ.
   - Update L: $[L \mapsto (A : a + a', B : b + b')]_{A,B}$
4. Now the sub-channel χ has been defunded, it can be safely discarded.

2.1.3 Topping up a ledger channel

Here we show how a participant can increase their funds held in a ledger channel by depositing into it. They can do this without disturbing any sub-channels supported by the ledger channel.

1. In this process A wants to deposit an additional a' coins into the the ledger channel L. We start in the state where L contains balances for A and B, as well as funding a sub-channel, χ:
   - $[\![L{:}x]\!]$, $[L \mapsto (A : a, B : b, \chi : c)]_{A,B}$, where $x = a + b + c$.
2. To prepare for the deposit the participants update the state to move A's entry to the end, simultaneously increasing A's total. This is a safe operation due to the precedence rules: as the channel is currently underfunded A would still only receive a if the outcome went to chain.
   - Update channel L: $[L \mapsto (B : b, \chi : c, A : a + a')]_{A,B}$
3. It is now safe for A to deposit into the channel on-chain:
   - Deposit by A: $D_L(a')[\![L{:}x]\!] = [\![L{:}x + a']\!]$
4. Finally, if required, the participants can reorder the state again:
   - Update channel L: $[L \mapsto (A : a + a', B : b, \chi : c)]_{A,B}$

2.1.4 Partial checkout from a ledger channel

A partial checkout is the opposite of a top up: one participant has excess funds in the ledger channel that they wish to withdraw on-chain. The participants want to do this without disturbing any sub-channels supported by the ledger channels.

1. We start with a ledger channel, L, that A wants to withdraw a' coins from:
   - $[\![L{:}x]\!]$, $[L \mapsto (A : a + a', B : b, \chi : c)]_{A,B}$, where $x = a + a' + b + c$.
2. The participants start by creating a new ledger channel, L', whose state reflects the situation they want to be in after A has withdrawn their coins. This is safe to do as this channel is currently unfunded.
   - Create channel L: $[L' \mapsto (A : a, B : b, \chi : c)]_{A,B}$
3. They then update L to fund L' alongside the coins that A wants to withdraw. They conclude the channel in this state:
   - Update channel L: $[L \mapsto (L' : a + b + c, A : a')]_{A,B}$
4. They then finalize the outcome of L on-chain. This can be done without waiting the timeout, assuming they both signed the conclusion proof in the previous step:
   - Finalize L on-chain: $[\![L{:}x \mapsto (L' : a + b + c, A : a')]\!]$
5. A can then call the transfer operation to get their coins under their control. They can optionally move the coins into L' at the same time:
   - Transfer coins to A: $T_{L,A}(a')[\![L{:}x \mapsto (L : a + b + c, A : a')]\!] = [\![L{:}x - a' \mapsto (L' : a + b + c),\ A{:}a]\!]$
   - Transfer coins to L': $T_{L,L'}(a + b + c)[\![L{:}x \mapsto (L : a + b + c),\ A{:}a']\!] = [\![L'{:}a + b + c,\ A{:}a]\!]$

Note that A was able to withdraw their funds instantly, without having to wait for the channel timeout.

3 Nitro

Nitro protocol is an extension to Turbo protocol. In Nitro protocol, the outcome of a channel can be either an allocation or a **guarantee**. A guarantee outcome specifies a target allocation, whose debts it will help to pay. When paying out debts, the guarantee outcome can choose to modify the payout priority order of its target allocation.

We will use the notation $(B|C, D)$ for a guarantee with target B, which prioritizes first C, then D, then to any other addresses according to the priorities

in B's allocation. We say a guarantee channel, A, which targets an allocation channel, B, 'can afford x for C', if C would receive at least x coins, were the coins currently held in A to be paid out according to A's reprioritization of B's allocation.

Nitro adds the **claim** operation, $C_{A,C}(x)$, to the existing transfer, deposit and withdraw operations. If A acts as guarantor for B and can afford x for C, then $C_{A,C}(x)$: (a) reduces the funds held by A by x, (b) increases the funds held by C by x, and (c) reduces the amount owed to C in the outcome of B. If the outcome of B is not yet registered on-chain, or if A cannot afford x for C, then the operation has no effect.

Example 3.1. In the following example, we have a guarantee channel, G, which holds 5 coins and guarantees L's allocation, with χ as highest priority.

$$C_{G,\chi}(2)[\![G{:}5 \mapsto (L|A), L{:}(A:5, \chi:5)]\!] = \\ [\![G{:}3 \mapsto (L|A), L{:}(A:5, \chi:3), \chi{:}4]\!]$$

Note that after the claim has gone through, L's debt to χ has decreased.

We give a python implementation of the Nitro adjudicator in the appendix.

3.1 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, C. Our starting point for this channel is a pair of ledger channels, L and L', with participants $\{A, C\}$ and $\{B, C\}$ respectively.

$$[\![L{:}x, L'{:}x]\!],\ [L \mapsto (A:a, C:b)]_{A,C},\ [L' \mapsto (B:b, C:a)]_{B,C} \tag{1}$$

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 $\{A, B, C\}$ and two guarantor channels G and G' which target J. The setup is shown in figure 1.

We will cover the steps for safely setting up this construction in section 3.2. In the rest of this section we will explain why this construction can be considered to fund the channel χ. Similarly to the method for ledger channel construction, we will do this by demonstrating how any one of the participants can offload the channel χ: converting it to an on-chain channel that holds its own funds.

We will first consider the case where A wishes to offload χ. We will assume that A starts by finalizing all their finalizable channels on-chain, followed by a

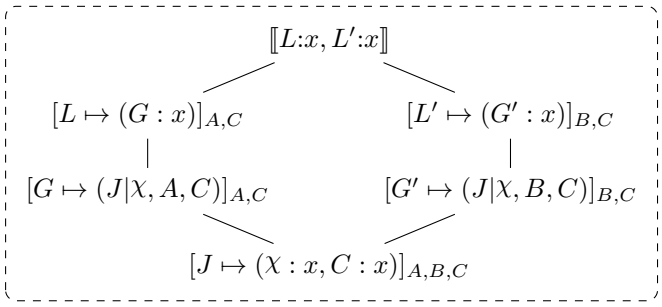

Figure 1: Virtual channel construction

transferring funds held in the ledger channel L to the guarantor channel G. The final step is for A to claim on G's guarantee for χ:

$$C_{G,\chi}(x)[\![G{:}x \mapsto (J|\chi, A, C), L'{:}x, J \mapsto (\chi : x, C : x)]\!] = \\ [\![L'{:}x, \chi{:}x, J \mapsto (C : x)]\!] \quad (2)$$

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 C wants to offload the channel and reclaim their funds. This is important to ensure that A and B cannot lock C's funds indefinitely in the channel. We assume that C has started by registering all their finalizable channels on-chain, followed by transferring funds from L to G and from L' to G'.

$$C_{G',G}(C)C_{G,\chi}(x)[\![G{:}x \mapsto (J|\chi, A, C), G'{:}x \mapsto (J|\chi, B, C), \\ J \mapsto (\chi : x, C : x)]\!] = [\![L'{:}x, \chi{:}x, C{:}x]\!] \quad (3)$$

Note that C has to claim on both guarantees, offloading χ before being able to reclaim their funds.

As in Turbo, the offload is an action of last resort and virtual channels are designed to be opened and closed entirely off-chain. We now show how this can be accomplished.

3.2 Opening and Closing Virtual Channels

In this section we present a sequence of system 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 system remains unchanged throughout for all participants involved. We justify this claim in the appendix.

The procedure for opening a virtual channel is as follows:

1. Start in the state given in equation (1):
   - $[\![L{:}x, L'{:}x]\!]$, $[L \mapsto (A : a, C : b)]_{A,C}$, $[L' \mapsto (B : b, C : a)]_{B,C}$
2. A and B bring their channel χ to the funding point:
   - Create channel χ: $[\chi \mapsto (A : a, B : b)]_{A,B}$
3. In any order, A, B and C setup the virtual channel construction:
   - Create channel J: $[J \mapsto (A : a, B : b, C : c)]_{A,B,C}$
   - Create channel G: $[G \mapsto (J|\chi, A, C)]_{A,C}$
   - Create channel G': $[G' \mapsto (J|\chi, B, C)]_{B,C}$
4. In either order switch the ledger channels over to fund the guarantees:
   - Update L: $[L \mapsto (G : x)]_{A,C}$
   - Update L': $[L' \mapsto (G' : x)]_{B,C}$
5. Switch J over to fund χ:
   - Update channel J: $[J \mapsto (\chi : x, C : x)]_{A,B,C}$

We give a visual representation of this procedure in figure 2.

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:
   - Update χ: $[\chi \mapsto (A : a', B : b')]_{A,B}$
2. A and B sign an update to J to take account of the outcome of χ. C will accept this update, provided that their allocation of x coins remains the same:
   - Update channel J: $[J \mapsto (A : a', B : b', C : x)]_{A,B,C}$
3. In either order switch the ledger channels to absorb the outcome of J, defunding the guarantor channels in the process:
   - Update L: $[L \mapsto (A : a', C : b')]_{A,C}$
   - Update L': $[L' \mapsto (B : b', C : a')]_{B,C}$
4. The channels χ, J, G and G' are now all defunded, so can be discarded

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

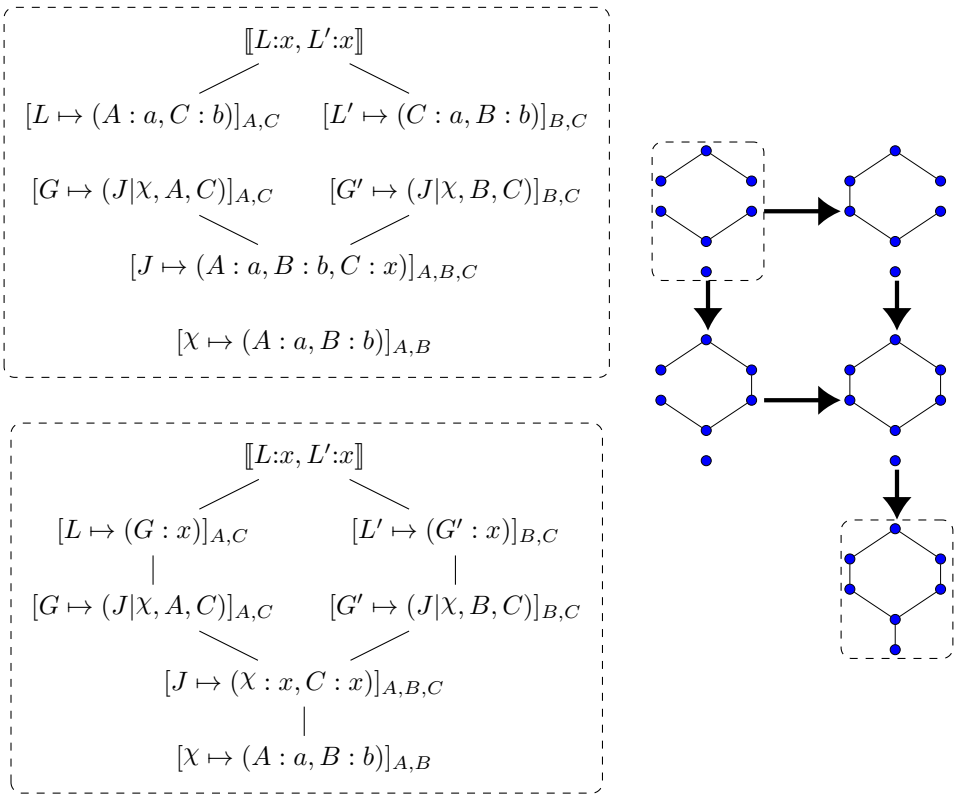

Figure 2: Opening a virtual channel