---

title: State Channels description: An introduction to state channels and payment channels as a scaling solution currently utilized by the Ethereum community. lang: en sidebarDepth: 3

State channels allow participants to securely transact off-chain while keeping interaction with Ethereum Mainnet at a minimum. Channel peers can conduct an arbitrary number of off-chain transactions while only submitting two on-chain transactions to open and close the channel. This allows for extremely high transaction throughput and results in lower costs for users.

Prerequisites {#prerequisites}

You should have read and understood our pages on Ethereum scaling and layer 2.

What are channels? {#what-are-channels}

Public blockchains, such as Ethereum, face scalability challenges due to their distributed architecture: on-chain transactions must be executed by all nodes. Nodes have to be able to handle the volume of transactions in a block using modest hardware, imposing a limit on the transaction throughput to keep the network decentralized. Blockchain channels solve this problem by allowing users to interact off-chain while still relying on the security of the main chain for final settlement.

Channels are simple peer-to-peer protocols that allow two parties to make many transactions between themselves and then only post the final results to the blockchain. The channel uses cryptography to demonstrate that the summary data they generate is truly the result of a valid set of intermediate transactions. A "multisig" smart contract ensures the transactions are signed by the correct parties.

With channels, state changes are executed and validated by interested parties, minimizing computation on Ethereum's execution layer. This decreases congestion on Ethereum and also increases transaction processing speeds for users.

Each channel is managed by a multisig smart contract running on Ethereum. To open a channel, participants deploy the channel contract on-chain and deposit funds into it. Both parties collectively sign a state update to initialize the channel's state, after which they can transact quickly and freely off-chain.

To close the channel, participants submit the last agreed-upon state of the channel on-chain. Afterward, the smart contract distributes the locked funds according to each participant's balance in the channel's final state.

Peer-to-peer channels are particularly useful for situations where some predefined participants wish to transact with high frequency without incurring visible overhead. Blockchain channels fall under two categories: **payment channels** and **state channels**.

Payment channels {#payment-channels}

A payment channel is best described as a "two-way ledger" collectively maintained by two users. The ledger's initial balance is the sum of deposits locked into the on-chain contract during the channel opening phase. Payment channel transfers can be performed instantaneously and without the involvement of the actual blockchain itself, except for an initial one-time on-chain creation and an eventual closing of the channel.

Updates to the ledger's balance (i.e., the payment channel's state) require the approval of all parties in the channel. A channel update, signed by all channel participants, is considered finalized, much like a transaction on Ethereum.

Payment channels were among the earliest scaling solutions designed to minimize expensive on-chain activity of simple user interactions (e.g. ETH transfers, atomic swaps, micropayments). Channel participants can conduct an unlimited amount of instant, feeless transactions between each other as long as the net sum of their transfers does not exceed the deposited tokens.

State channels {#state-channels}

Apart from supporting off-chain payments, payment channels have not proven useful for handling general state transition logic. State channels were created to solve this problem and make channels useful for scaling general-purpose computation.

State channels still have a lot in common with payment channels. For example, users interact by exchanging cryptographically signed messages (transactions), which the other channel participants must also sign. If a proposed state update isn't signed by all participants, it is considered invalid.

However, in addition to holding the user's balances, the channel also tracks the current state of the contract's storage (i.e., values of contract variables).

This makes it possible to execute a smart contract off-chain between two users. In this scenario, updates to the smart contract's internal state require only the approval of the peers who created the channel.

While this solves the scalability problem described earlier, it has implications for security. On Ethereum, the validity of state transitions on Ethereum is enforced by the network's consensus protocol. This makes it impossible to propose an invalid update to a smart contract's state or alter smart contract execution.

State channels don't have the same security guarantees. To some extent, a state channel is a miniature version of Mainnet. With a limited set of participants enforcing rules, the possibility of malicious behavior (e.g., proposing invalid state updates) increases. State channels derive their security from a dispute arbitration system based on fraud proofs.

How state channels work {#how-state-channels-work}

Basically, the activity in a state channel is a session of interactions involving users and a blockchain system. Users mostly communicate with each other off-chain and only interact with the underlying blockchain to open the channel, close the channel, or settle potential disputes between participants.

The following section outlines the basic workflow of a state channel:

Opening the channel {#opening-the-channel}

Opening a channel requires participants to commit funds to a smart contract on Mainnet. The deposit also functions as a virtual tab, so participating actors can transact freely without needing to settle payments immediately. Only when the channel is finalized on-chain do parties settle each other and withdraw what's left of their tab.

This deposit also serves as a bond to guarantee honest behavior from each participant. If depositors are found guilty of malicious actions during the dispute resolution phase, the contract slashes their deposit.

Channel peers must sign an initial state, which they all agree upon. This serves as the state channel's genesis, after which users can start transacting.

Using the channel {#using-the-channel}

After initializing the channel's state, peers interact by signing transactions and sending them to each other for approval. Participants initiate state updates with these transactions and sign state updates from others. Each transaction comprises the following:

- A **nonce**, which acts as a unique ID for transactions and prevents replay attacks. It also identifies the order in which state updates occurred (which is important for dispute resolution)
- The channel's old state
- The channel's new state
- The transaction which triggers the state transition (e.g., Alice sends 5 ETH to Bob)

State updates in the channel are not broadcasted on-chain as is normally the case when users interact on Mainnet, which aligns with state channels' goal to minimize on-chain footprint. As long as participants agree on state updates, they are as final as an Ethereum transaction. Participants only need to depend on Mainnet's consensus if a dispute arises.

Closing the channel {#closing-the-channel}

Closing a state channel requires submitting the channel's final, agreed-upon state to the on-chain smart contract. Details referenced in the state update include the number of each participant's moves and a list of approved transactions.

After verifying that the state update is valid (i.e., it is signed by all parties) the smart contract finalizes the channel and distributes the locked funds according to the channel's outcome. Payments made off-chain are applied to Ethereum's state and each participant receives their remaining portion of the locked funds.

The scenario described above represents what happens in the happy case. Sometimes, users may be unable to reach an agreement and finalize the channel (the sad case). Any of the following could be true of the situation:

- Participants go offline and fail to propose state transitions
- Participants refuse to co-sign valid state updates
- Participants try to finalize the channel by proposing an old state update to the on-chain contract
- Participants propose invalid state transitions for others to sign

Whenever consensus breaks down between participating actors in a channel, the last option is to rely on Mainnet's consensus to enforce the channel's final, valid state. In this case, closing the state channel requires settling disputes on-chain.

Settling disputes {#settling-disputes}

Typically, parties in a channel agree on closing the channel beforehand and co-sign the last state transition, which they submit to the smart contract. Once the update is approved on-chain, execution of the off-chain smart contract ends and participants exit the channel with their money.

However, one party can submit an on-chain request to end the smart contract's execution and finalize the channel—without waiting for their counterpart's approval. If any of the consensus-breaking situations described earlier occur, either party can trigger the on-chain contract to close the channel and distribute funds. This provides **trustlessness**, ensuring that honest parties can exit their deposits at any point, regardless of the other party's actions.

To process the channel exit, the user must submit the application's last valid state update to the on-chain contract. If this checks out (i.e., it bears the signature of all parties), then funds are redistributed in their favor.

There is, however, a delay in executing single-user exit requests. If the request to conclude the channel was unanimously approved, then the on-chain exit transaction is executed immediately.

The delay comes into play in single-user exits due to the possibility of fraudulent actions. For example, a channel participant may try to finalize the channel on Ethereum by submitting an older state update on-chain.

As a countermeasure, state channels allow honest users to challenge invalid state updates by submitting the latest, valid state of the channel on-chain. State channels are designed such that newer, agreed-upon state updates trump older state updates.

Once a peer triggers the on-chain dispute-resolution system, the other party is required to respond within a time limit (called the challenge window). This allows users to challenge the exit transaction, especially if the other party is applying a stale update.

Whatever the case may be, channel users always have strong finality guarantees: if the state transition in their possession was signed by all members and is the most recent update, then it is of equal finality with a regular on-chain transaction. They still have to challenge the other party on-chain, but the only possible outcome is finalizing the last valid state, which they hold.

How do state channels interact with Ethereum? {#how-do-state-channels-interact-with-ethereum}

Although they exist as off-chain protocols, state channels have an on-chain component: the smart contract deployed on Ethereum when opening the channel. This contract controls the assets deposited into the channel, verifies state updates, and arbitrates disputes between participants.

State channels don't publish transaction data or state commitments to Mainnet, unlikelayer 2 scaling solutions. However, they are more connected to Mainnet than, say, sidechains, making them somewhat safer.

State channels rely on the main Ethereum protocol for the following:

1. Liveness {#liveness}

The on-chain contract deployed when opening the channel is responsible for the channel's functionality. If the contract is running on Ethereum, then the channel is always available for usage. Conversely, a sidechain can always fail, even if Mainnet is operational, putting user funds at risk.

2. Security {#security}

To some extent, state channels rely on Ethereum to provide security and protect users from malicious peers. As discussed in later sections, channels use a fraud proof mechanism that lets users challenge attempts to finalize the channel with an invalid or stale update.

In this case, the honest party provides the latest valid state of the channel as a fraud proof to the on-chain contract for verification. Fraud proofs enable mutually distrustful parties to conduct off-chain transactions without risking their funds in the process.

3. Finality {#finality}

State updates collectively signed by channel users are considered as good as on-chain transactions. Still, all in-channel activity only achieves true finality when the channel is closed on Ethereum.

In the optimistic case, both parties can cooperate and sign the final state update and submit on-chain to close the channel, after which the funds are distributed according to the channel's final state. In the pessimistic case, where someone tries to cheat by posting an incorrect state update on-chain, their transaction isn't finalized until the challenge window elapses.

Virtual state channels {#virtual-state-channels}

The naive implementation of a state channel would be to deploy a new contract when two users wish to execute an application off-chain. This is not only infeasible, but it also negates the cost-effectiveness of state channels (on-chain transaction costs can quickly add up).

To solve this problem, "virtual channels" were created. Unlike regular channels that require on-chain transactions to open and terminate, a virtual channel can be opened, executed, and finalized without interacting with the main chain. It is even possible to settle disputes off-chain using this method.

This system relies on the existence of so-called "ledger channels", which have been funded on-chain. Virtual channels between two parties can be built on top of an existing ledger channel, with the owner(s) of the ledger channel serving as an intermediary.

Users in each virtual channel interact via a new contract instance, with the ledger channel able to support multiple contract instances. The ledger channel's state also contains more than one contract storage state, allowing for parallel execution of applications off-chain between different users.

Just like regular channels, users exchange state updates to progress the state machine. Except a dispute arises, the intermediary only has to be contacted when opening or terminating the channel.

Virtual payment channels {#virtual-payment-channels}

Virtual payment channels work off the same idea as virtual state channels: participants connected to the same network can

pass messages without needing to open a new channel on-chain. In virtual payment channels, value transfers are routed through one or more intermediaries, with guarantees that only the intended recipient can receive transferred funds.

Applications of state channels {#applications-of-state-channels}

Payments {#payments}

Early blockchain channels were simple protocols that allowed two participants to conduct rapid, low-fee transfers off-chain without having to pay high transaction fees on Mainnet. Today, payment channels are still useful for applications designed for the exchange and deposits of ether and tokens.

Channel-based payments have the following advantages:

1. **Throughput**: The amount of off-chain transactions per channel is unconnected to Ethereum's throughput, which is influenced by various factors, especially block size and block time. By executing transactions off-chain, blockchain channels can achieve higher throughput.
2. **Privacy**: Because channels exist off-chain, details of interactions between participants are not recorded on Ethereum's public blockchain. Channel users only have to interact on-chain when funding and closing channels or settling disputes. Thus, channels are useful for individuals who desire more private transactions.
3. **Latency**: Off-chain transactions conducted between channel participants can be settled instantly, if both parties cooperate, reducing delays. In contrast, sending a transaction on Mainnet requires waiting for nodes to process the transaction, produce a new block with the transaction, and reach consensus. Users may also need to wait for more block confirmations before considering a transaction finalized.
4. **Cost**: State channels are particularly useful in situations where a set of participants will exchange many state updates over a long period. The only costs incurred are the opening and closing of the state channel smart contract; every state change between opening and closing the channel will be cheaper than the last as the settlement cost is distributed accordingly.

Implementing state channels on layer 2 solutions, such as rollups, could make them even more attractive for payments. While channels offer cheap payments, the costs of setting up the on-chain contract on Mainnet during the opening phase can be get expensive—especially when gas fees spike. Ethereum-based rollups offer lower transaction fees and can reduce overhead for channel participants by bringing down setup fees.

Microtransactions {#microtransactions}

Microtransactions are low-value payments (e.g., lower than a fraction of a dollar) that businesses cannot process without incurring losses. These entities must pay payment service providers, which they cannot do if the margin on customer payments is too low to make a profit.

Payment channels solve this problem by reducing the overhead associated with microtransactions. For example, an Internet Service Provider (ISP) can open a payment channel with a customer, allowing them to stream small payments each time they use the service.

Beyond the cost of opening and closing the channel, participants don't incur further costs on microtransactions (no gas fees). This is a win-win situation since customers have more flexibility in how much they pay for services and businesses don't lose out on profitable microtransactions.

Decentralized applications {#decentralized-applications}

Like payment channels, state channels can make conditional payments according to the state machine's final states. State channels can also support arbitrary state transition logic, making them useful for executing generic apps off-chain.

State channels are often limited to simple turn-based applications, as this makes it easier to manage funds committed to the on-chain contract. Also, with a limited number of parties updating the off-chain application's state at intervals, punishing dishonest behavior is relatively straightforward.

The efficiency of a state channel application also depends on its design. For example, a developer might deploy the app channel contract on-chain once and allow other players to re-use the app without having to go on-chain. In this case, the initial app channel serves as a ledger channel supporting multiple virtual channels, each running a new instance of the app's smart contract off-chain.

A potential use-case for state channel applications is simple two-player games, where funds are distributed based on the game's outcome. The benefit here is that players don't have to trust each other (trustlessness) and the on-chain contract, not players, controls the allocation of funds and settlement of disputes (decentralization).

Other possible use-cases for state channel apps include ENS name ownership, NFT ledgers, and many more.

Atomic transfers {#atomic-transfers}

Early payment channels were restricted to transfers between two parties, limiting their usability. However, the introduction of virtual channels allowed individuals to route transfers through intermediaries (i.e., multiple p2p channels) without having to open a new channel on-chain.

Commonly described as "multi-hop transfers", routed payments are atomic (i.e., either all parts of the transaction succeed or it fails altogether). Atomic transfers use Hashed Timelock Contracts (HTLCs) to ensure the payment is released only if certain conditions are met, thereby reducing counterparty risk.

Drawbacks of using state channels {#drawbacks-of-state-channels}

Liveness assumptions {#liveness-assumptions}

To ensure efficiency, state channels place time limits on the ability of channel participants to respond to disputes. This rule assumes that peers will always be online to monitor channel activity and contest challenges when necessary.

In reality, users can go offline for reasons out of their control (e.g., poor internet connection, mechanical failure, etc.). If an honest user goes offline, a malicious peer can exploit the situation by presenting old intermediate states to the adjudicator contract and stealing the committed funds.

Some channels use "watchtowers"—entities responsible for watching on-chain dispute events on behalf of others and taking necessary actions, like alerting concerned parties. However, this can add to the costs of using a state channel.

Data unavailability {#data-unavailability}

As explained earlier, challenging an invalid dispute requires presenting the latest, valid state of the state channel. This is another rule based on an assumption—that users have access to the channel's latest state.

Although expecting channel users to store copies of off-chain application state is reasonable, this data may be lost due to error or mechanical failure. If the user doesn't have the data backed up, they can only hope that the other party doesn't finalize an invalid exit request using old state transitions in their possession.

Ethereum users don't have to deal with this problem since the network enforces rules on data availability. Transaction data is stored and propagated by all nodes and available for users to download if and when necessary.

Liquidity issues {#liquidity-issues}

To establish a blockchain channel, participants need to lock funds in an on-chain smart contract for the channel's lifecycle. This reduces the liquidity of channel users and also limits channels to those who can afford to keep funds locked on Mainnet.

However, ledger channels—operated by an off-chain service provider (OSP)—can reduce liquidity issues for users. Two peers connected to a ledger channel can create a virtual channel, which they can open and finalize completely off-chain, anytime they want.

Off-chain service providers could also open channels with multiple peers, making them useful for routing payments. Of

course, users must pay fees to OSPs for their services, which may be undesirable for some.

Griefing attacks {#griefing-attacks}

Griefing attacks are a common feature of fraud proof-based systems. A griefing attack does not directly benefit the attacker but causes grief (i.e., harm) to the victim, hence the name.

Fraud proving is susceptible to griefing attacks because the honest party must respond to every dispute, even invalid ones, or risk losing their funds. A malicious participant can decide to repeatedly post stale state transitions on-chain, forcing the honest party to respond with the valid state. The cost of those on-chain transactions can quickly add up, causing honest parties to lose out in the process.

Predefined participant sets {#predefined-participant-sets}

By design, the number of participants that comprise a state channel remains fixed throughout its lifetime. This is because updating the participant set would complicate the channel's operation, especially when funding the channel, or settling disputes. Adding or removing participants would also require additional on-chain activity, which increases overhead for users.

While this makes state channels easier to reason about, it limits the usefulness of channel designs to application developers. This partly explains why state channels have been dropped in favor of other scaling solutions, such as rollups.

Parallel transaction processing {#parallel-transaction-processing}

Participants in the state channel send state updates in turns, which is why they work best for "turn-based applications" (e.g., a two-player chess game). This eliminates the need to handle simultaneous state updates and reduces the work the on-chain contract must do to punish stale update posters. However, a side-effect of this design is that transactions are dependent on each other, increasing latency and diminishing the overall user experience.

Some state channels solve this problem by using a "full-duplex" design that separates the off-chain state into two unidirectional "simplex" states, allowing for concurrent state updates. Such designs improve off-chain throughput and decrease transaction delays.

Use state channels {#use-state-channels}

Multiple projects provide implementations of state channels that you can integrate into your dapps:

- Connext
- Kchannels
- Perun
- Raiden
- Statechannels.org

Further reading {#further-reading}

State channels

- Making Sense of Ethereum's Layer 2 Scaling Solutions: State Channels, Plasma, and Truebit – *Josh Stark, Feb 12 2018*
- State Channels - an explanation *Nov 6, 2015 - Jeff Coleman*
- Basics of State Channels *District0x*
- Blockchain State Channels: A State of the Art

Know of a community resource that helped you? Edit this page and add it!