Foundations of Blockchain

Matteo Nardelli

April 13, 2026

Scaling Blockchains

On-chain versus Off-chain

- A payment is on-chain if it is recorded as a transaction on the blockchain;
- Off-chain payments are not visible in the underlying blockchain. Usually, these payments are privately exchanged between the involved parties.

Exchanging payments on-chain requires submitting a transaction and waiting its inclusion in a block. However, blockchains have limited throughput performance:

- Bitcoin: 7 transactions per second (TPS), a block every 10 min;
- Ethereum: 12–15 TPS, a block every 12 s;
- Algorand: 8–12 TPS (but theoretically, up to 7500 TPS), a new block every 3.9 s;
- Different factors can affect these numbers (e.g., block size, fees, consensus).

Scaling Blockchains (1)

To mitigate the performance bottleneck, many approaches have been proposed:

- **DAG**-based blockchains;
- **Sharding technique**: splits transactions into shards, which are processed in parallel; however, it is hard to achieve consensus across shards;
- **Layer-2 protocols**: process certain transactions outside of the main chain, but the consensus of theses transactions relies on a parent chain;
- **Sidechain** technique: separate (auxiliary) blockchain that processes transactions individually. It has its own consensus algorithm. It interacts with the main chain.
- **Heterogeneous structure**: uses different types of block (e.g., *keyblocks* to conduct consensus; *microblocks* to vote for leaders and carry transactions).

Scaling Blockchains (2)

- **Hybrid consensus**: 2+ consensus protocols, e.g., to smoothly switch between optimistic conditions (PBFT) and the worst-case conditions (PoW).

Wang et al, "SoK: DAG-based Blockchain Systems". ACM Comput. Surv. 55(12), art. 261 (2023).

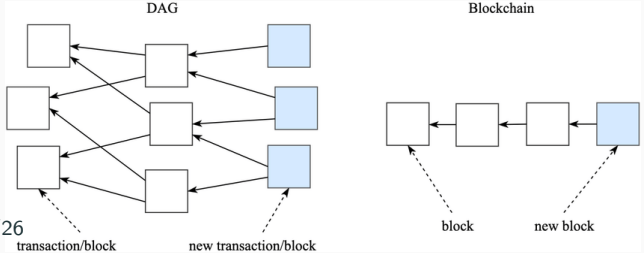
Scaling Blockchains

DAG-based Blockchains

DAG-based Blockchains

Directed Acyclic Graph (DAG)-based Blockchains:

- Blockchains maintain transactions/blocks in one single chain.
  - Concurrent transactions/blocks compete for one valid position each round.
  - Leading to slow confirmation.
- DAG-based blockchains structure transactions/blocks in the form of graph;
- They can improve **performance** by requiring less communication, computation, and storage overhead.



DAG-based Blockchains

Rough classifications of DAG-based Blockchains:

- Unit Representation:
  1. Requests are immediately handled whenever received;
  2. Requests packaged by powerful parties (e.g., miner, validator) and then disseminated;

DAG-based Blockchains

Rough classifications of DAG-based Blockchains:

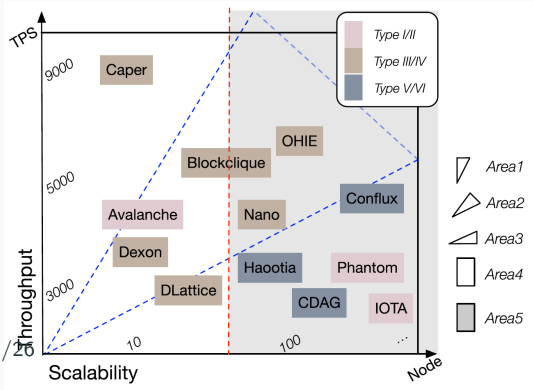
- Unit Representation:
  1. Requests are immediately handled whenever received;
  2. Requests packaged by powerful parties (e.g., miner, validator) and then disseminated;
- Graph Topology:
  1. Divergence: units sparsely spread in unpredictable directions without predetermined orders;
  2. Parallel: units are maintained in the form of multiple (parallel) chains;
  3. Convergence: units are organized in a determined sequence or tend to converge in a determined sequence.

Wang et al., "SoK: DAG-based Blockchain Systems", ACM Comput Surv 55, 12, Art. 261, 2023.

DAG-based Blockchains

	Divergence	*Parallel*	*Convergence*
1^{od}	IOTA [21], Graphchain [40], Avalanche [45] (*Type I*)	Nano [36], Caper [37], Vite [38], Chainweb [39], Hashgraph [41], DLattice [42], Aleph [43], Jointgraph [46], Lachesis [47–51] (*Type III*)	Byteball [22], Haootia [44] JHdag [26] (*Type V*)
2^{od}	Spectre [23], Phantom [56], Meshcash [63] (*Type II*)	Prism [52], Blockmania [53], Blockclique [54], OHIE [57], Sphinx [58, 59], Eunomia [60], DEXON [64], PARSEC [65] (*Type IV*)	GHOST [35], Inclusive [55], CDAG [61], Conflux [62], StreamNet [66] (*Type VI*)

DAG-based blockchain systems cannot improve the **performance**, **scalability**, **security**, **decentralization**, and **(strict) consistency** at the same time.



Scaling Blockchains

Modern Approaches: Layer-2

Scaling Blockchains: Recent Trends

- Key limits of blockchains:
  - Limited throughput ($\approx$ 10–100 TPS)
  - High latency (due to block time)
  - Varying costs (fees)
- Trade-off: scalability, security, decentralization (Trilemma)
- Recent trends:
  - Advanced layer-2 (rollup)
  - Modular blockchain design
  - Data availability separation
- Instead of scaling the blockchain, we build on top of it

From Off-chain to Layer-2

- First generation: Payment channels (lightning network)
- Second generation: State channels and Plasma
- Third generation: Rollup (for general-purpose computation)

Scaling Blockchains

The Lightning Network

The Lightning Network

- Bitcoin publicly records every transaction in a globally replicated ledger. Every transaction is seen, validated, and stored by every participating node;
- Once blocks are full, excess transactions are left to wait in a queue;
- Competition for fees can increase the cost of each transaction;
- Increase the block size limit implies utilization of more resources and may not completely solve the problem;
- Visa network processes (at peak) 40,000 TPS: Unlikely to scale a blockchain to validate the entire world's transactions in a decentralized way.

The Lightning Network

- Several efforts proposed to build payment channels aiming at processing the majority of transactions off-chain (e.g., [M+16][M+19][S+22]);
  - HTLC and multi-signatures used to bound layer-2 to layer-1;
  - Different approaches: payment channels, (optimistic and zero-knowledge) rollups;
  - Different abstractions: state channel, payment channel, payment channel hubs;
  - A survey on layer-2 protocols [G+23];
- The Lightning Network (LN) represents the most prominent solution for managing Payment Channel Networks (PCNs) [PD16]:
  - Several alternatives leverage the LN key ideas (e.g., Eclair[1], Raiden[2], Thunder[3]).

[1] https://github.com/ACINQ/eclair
[2] http://raiden.network/
[3] https://github.com/blockchain/thunder

The Lightning Network

- The Lightning Network (LN) is a second layer technology on top of Bitcoin.
- A matter of trust:
  - Cryptographic systems (like LN) allow to transact with people we do not trust;
  - This is not a trustless operation;
  - We still need trust in the used protocol (and its software implementation) that will result in fair outcomes;
  - Differently from traditional financial systems, cryptographic systems make trusted third parties unnecessary to ensure fair outcomes;

The Lightning Network

- The LN enables fast, secure, private, trustless, and permissionless payments:
  - **Fast:** Users can route payments to each other for low cost and in real time;
  - **Trustless:** Users who exchange value do not need to wait for block confirmations for payments;
  - **Secure:** Once a payment has completed, it is final and cannot be reversed.
  - **Privacy:** Payments are transmitted between pairs of nodes and are not visible to everyone, resulting in much greater privacy;
    - **Onion routing:** even the nodes involved in routing a payment are only directly aware of their predecessor and successor in the payment route;
  - **Resource-friendly:** Payments do not need to be stored permanently (fewer resources needed; hence, LN is cheaper);
  - **Safety:** LN uses real bitcoin, which is always in the possession and full control of the user.

Scaling Blockchains

Lightning Network: Payment Channels

Payment Channel

- A payment channel is a financial relationship between two nodes;
- It allocates a balance of funds and is managed by a cryptographic protocol;
- The cryptographic protocol consists in a 2-of-2 multisignature address:
  - 2-of-2 multisignature address: both parties hold a share to spend funds;
  - The protocol prevents either channel partner from spending the funds unilaterally (i.e., to cheat);
- The channel partners negotiate a sequence of transactions that spend from this multisignature address;
  - Instead of recording these transactions on the blockchain, *parties hold these transactions unspent*;
  - The latest transaction defines how that balance is divided between the parties;
  - As a new transaction is negotiated, the previous ones are revoked (neither party can regress to a previous state).

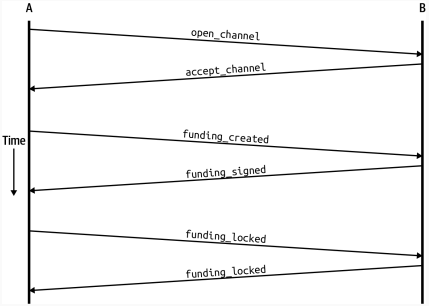
Payment Channel

- A payment on a payment channel is almost instant;
- If the channel is open, making a payment does not require the confirmation of Bitcoin blocks;
- Payments made in a payment channel are only known to the involved parties;
- Opening and closing channels requires an on-chain transaction:
  - This incurs transaction fees;
  - It is more convenient to keep channels open as long as possible.

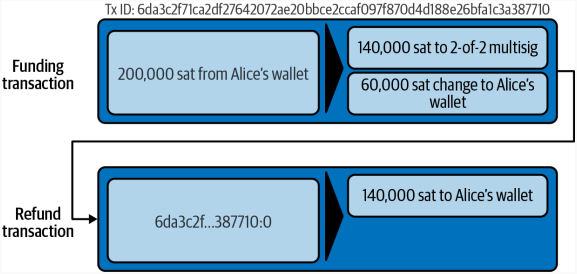
Funding Transaction

- One of the two channel partners will fund the payment channel by **sending bitcoin** to the 2-of-2 multisignature address;
- This transaction recorded on the Bitcoin blockchain;
  - The locking script (includes): `2 <PubKey1> <PubKey2> 2 CHECKMULTISIG`;
  - Later, we will detail the content of the funding transaction;
- The funding transaction is public, but it is not obvious that it is a Lightning payment channel;
  - It is a **P2SH** (Pay-to-Script-Hash), whose address always starts with 3;
- Channels are typically publicly announced by routing nodes that wish to forward payments (and earn from fees);
- **Private** channels (non-advertised) also exist;
  - e.g., by mobile nodes not participate in routing;

Funding Transaction



- `open_channel` and `accept_channel` exchange configuration values;
- `funding_create` creates the funding transaction, which is the signed by B; A requires B to sign also the refund transaction;

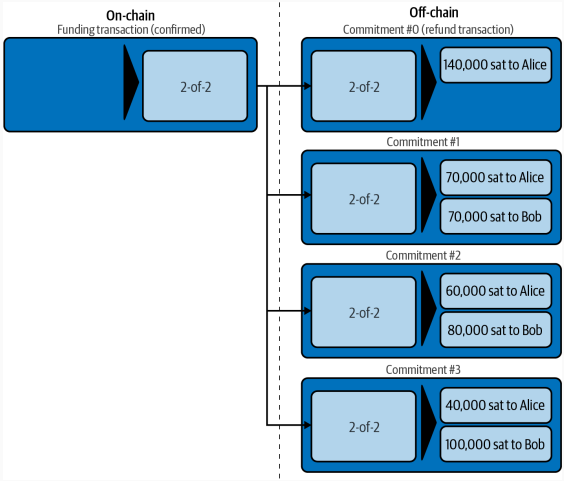


- When the funding transaction is confirmed on the blockchain, the parties exchange `funding_lock`

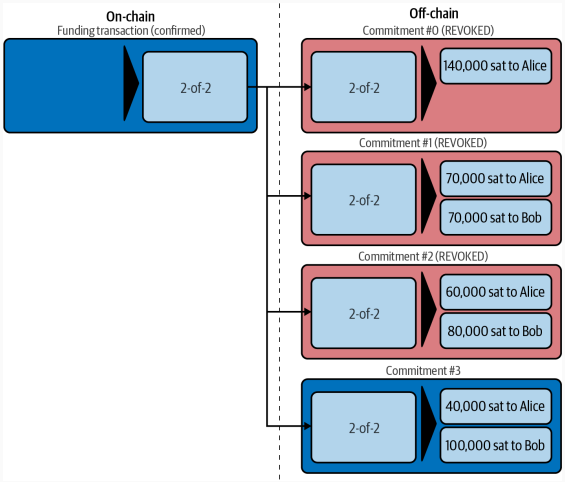
Commitment Transaction

After the funding transaction, commitment transactions are created each time the channel balance changes;

With a signed commitment transaction, each partner gives the other the ability to get his funds back;



Commitment Transaction

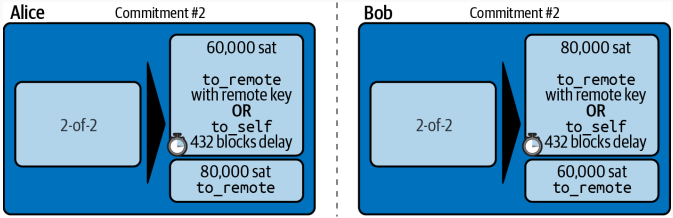


How to prevent publishing a previous commitment transaction?

- Commitment transactions are constructed so that if an old one is transmitted, the cheater can be punished.
- The penalty consists in giving the cheated party an opportunity to **claim the balance** of the cheater.

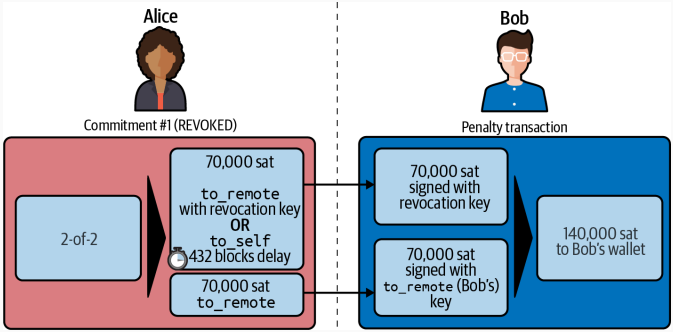
Commitment Transaction

- The commitment transaction includes:
  - a timelock delay: which prevents the owner from spending it immediately;
  - a revocation secret: which allows the other party to by-passing the timelock.
- The two channel partners hold two different variations of this transaction:



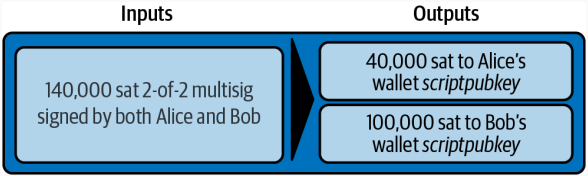
Commitment Transaction

- With a new commitment transaction, the previous revocation secret is revealed;
- If a party cheats, the other can immediately publish a penalty transaction to get its funds as well as the cheater's funds;



Closing a Channel

- Channel partners prefer not to close a channel (e.g., future use, avoid fees);
- However, sometimes it is necessary:
  - To reduce the balance held on Lightning channels (e.g., for security reasons);
  - The channel partner becomes unresponsive or not well-connected;
  - The channel partner has breached the protocol (i.e., closing is needed to protect funds);
- A mutual closing occurs by publishing a (co-signed) closing transaction with the last balance of the channel (it has no timelock);
  - Fees are paid by who opened the channel;



Closing a Channel

- A forced closing occurs by publishing the last **commitment transaction**;
- Forced closing is not recommended unless strictly necessary;
- Forced closing has higher fees:
  - The commitment transaction includes (up to five times) higher fee than the fee estimators suggest at the time the commitment transaction is negotiated;
  - The transaction includes additional outputs (e.g., time-lock, revocation hash);
  - Any pending routing attempts will have to be resolved on-chain;

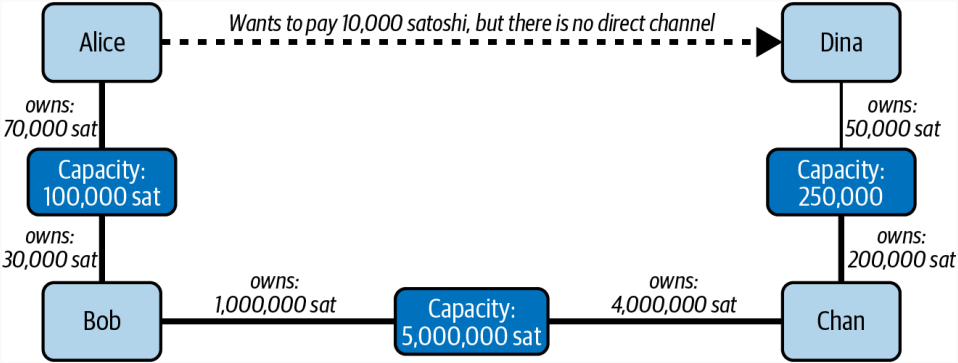
Scaling Blockchains

Lightning Network: Payment Channel Network

Routing

- The LN uses a gossip protocol to distribute public information about channels;
  - Not all information about a channel is propagated, to preserve privacy and scalability;
  - Propagated: capacity, channels partners;
  - Not propagated: balance, precise topology, single payments;
- When several participants have channels, payments can also be **forwarded** from channel to channel;
- The cryptographic protocol protects the entire network of participants:
  - They can forward payments without trusting any of the other participants;

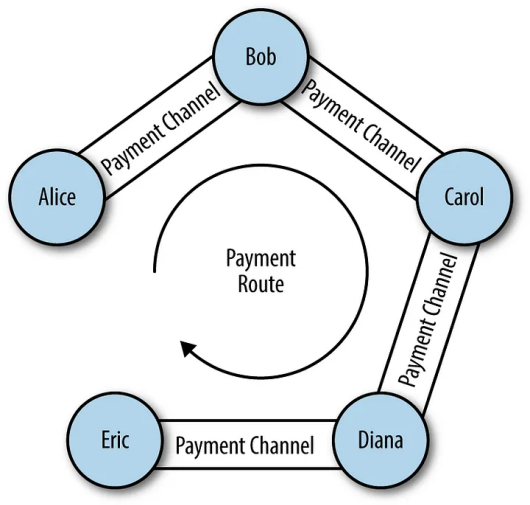
Payment Channel Network



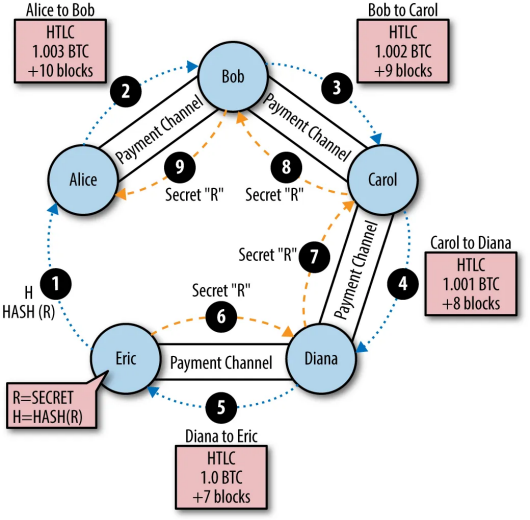
Payment Channel Network

- LN defines a fairness protocol, with the following properties:
  - **Trustless operation:** participants do not need to trust each other;
  - **Atomicity:** Either the payment is fully executed or it fails (everyone is refunded);
  - **Multihop:** The security extends end to end for payments routed through multiple payment channels.
  - (Optional) **Multipart Payments:** ability to split payments into multiple parts while maintaining atomicity
- Hash time-locked contract (HTLC):
  - Uses a cryptographic hash algorithm to commit to a randomly generated secret;
  - Conditions a payments to the knowledge of a value;
  - Uses a hash preimage as the secret that unlocks a payment;
  - Returns funds on timelock expiration;
- Alternative: Point Time-Locked Contract (PTLC):
  - Leverages properties of elliptic curves.

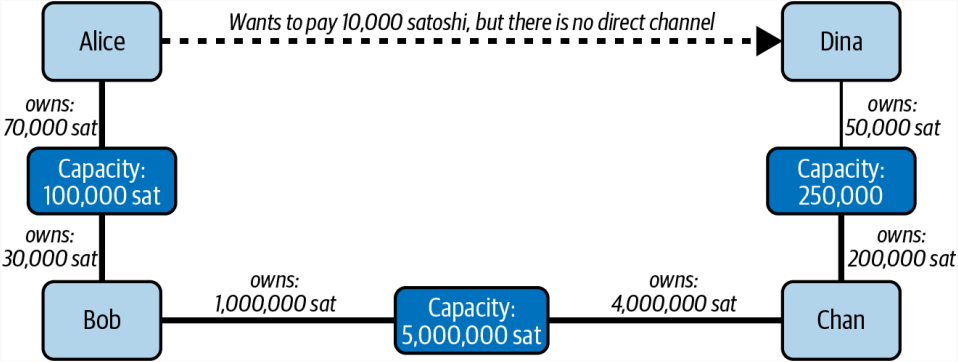
Payment Channel Network



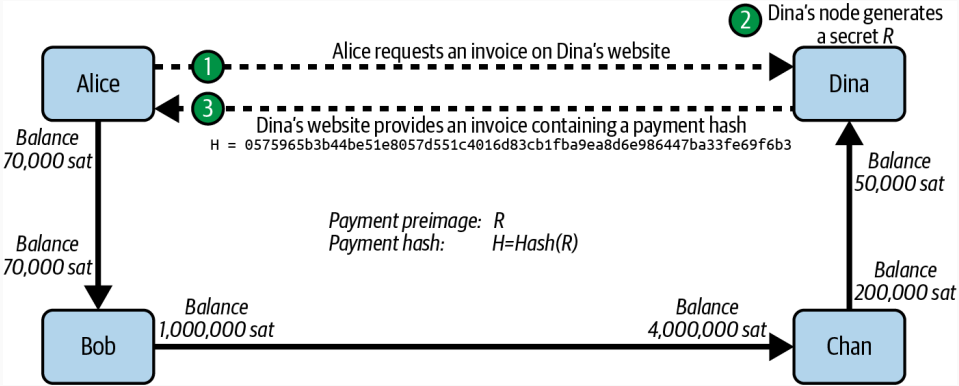
Payment Channel Network



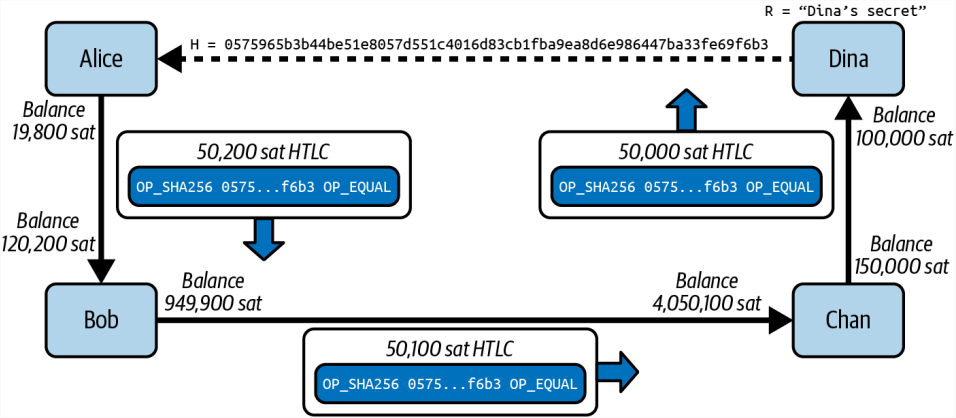
Payment Channel Network



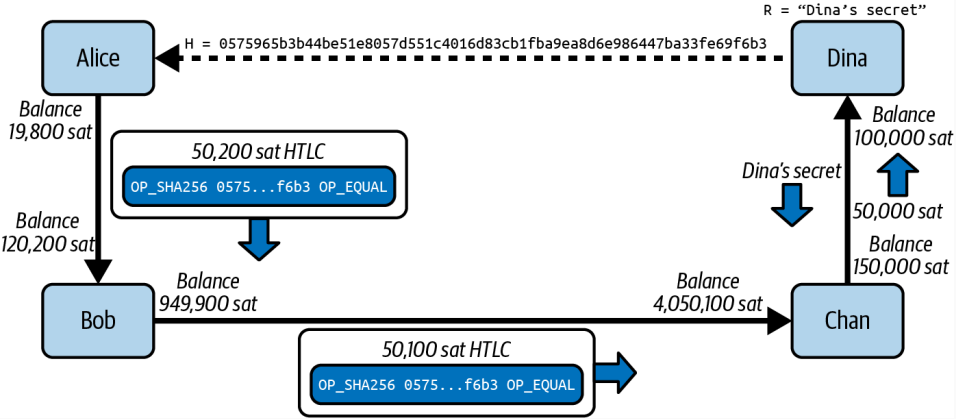
Payment Channel Network



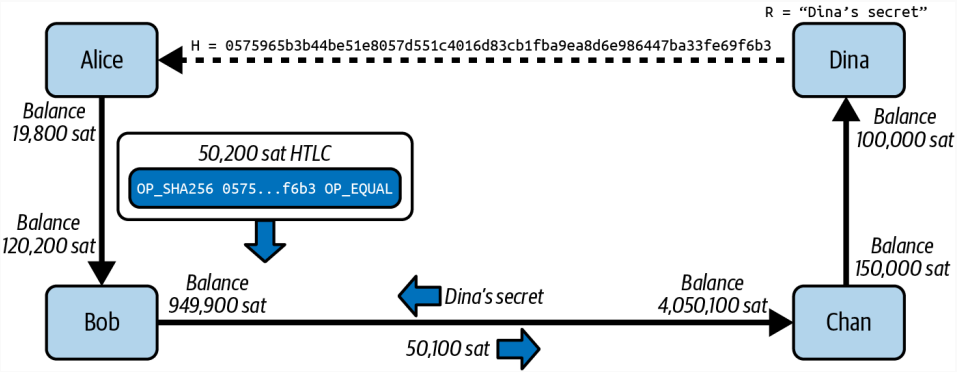
Payment Channel Network



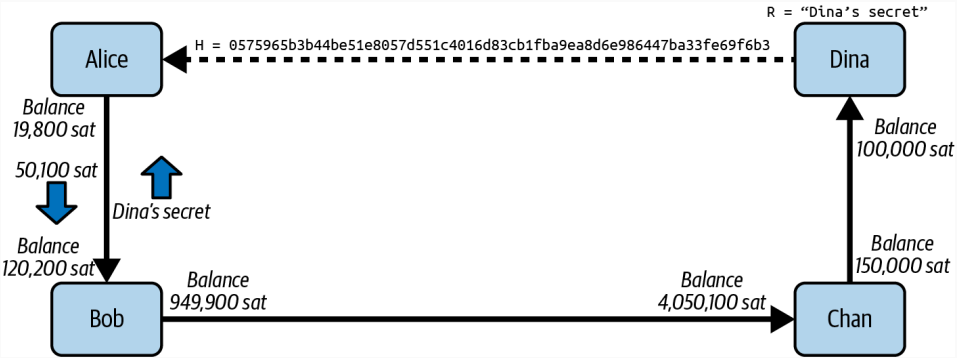
Payment Channel Network



Payment Channel Network



Payment Channel Network



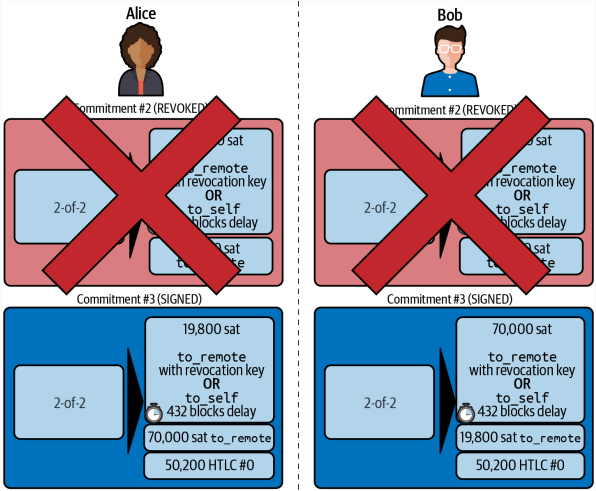
Payment Channel Network

Time Lock in HTLCs

- As HTLCs are extended from payer to payee, the time-locked refund clause in each HTLC has a **different time-lock value**;
- To ensure an orderly unwinding of a payment that fails, each hop needs to wait a bit less for their refund.
- For example, Alice sets the refund timelock to a block height of +500 blocks; Bob would then set the timelock to current + 450 blocks; Chan to current + 400 blocks.
- The decrementing timelock prevents race conditions and ensures the HTLC chain is unwound backward, from the destination toward the origin.

HTLC and Commitment Transaction

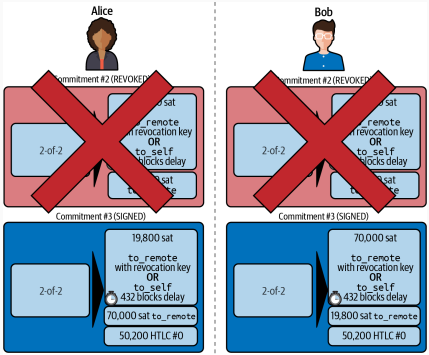
When B receives an HTLC from A, he creates a new commitment transaction with the same two outputs as before and a **new** one representing the HTLC;



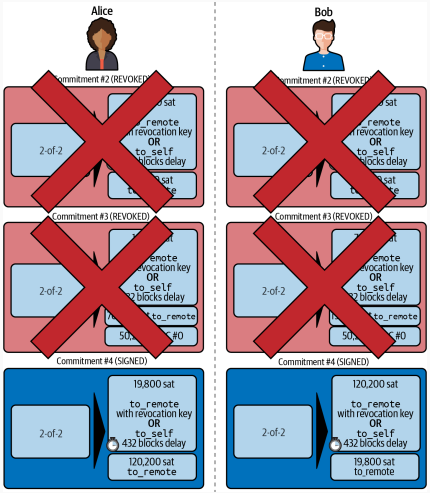
Multiple HTLCs in a Commitment Transaction

- A and B can have many HTLCs across a single channel;
- Each HTLC is added to the transaction as an **additional output**;
- At most 483 (pending) HTLCs are allowed on a channel;
  - This limit is imposed by the maximum Bitcoin transaction size;
- When the HTLC **preimage** is revealed and the payment is executed correctly, both A and B can remove the HTLC from the commitment transactions and update their channel balances.

Multiple HTLCs in a Commitment Transaction



Multiple HTLCs in a Commitment Transaction



Scaling Blockchains

Lightning Network: Routing

Source-based Path Finding

- The exact channel balances of every channel is unknown;
- Multiple path-finding and routing algorithms can coexist on the LN;
  - Path Finding: the process of finding and choosing a contiguous path made of payment channels that connects sender A to recipient B.
  - Routing: the active process of sending a payment on a path, which involves the cooperation of all the intermediary nodes along that path.
- Source-based path-finding is successful at the current scale of LN;
  - The path-finding strategy currently implemented is to **iteratively try** paths until one is found that has enough liquidity to forward the payment.
  - This does not necessarily result in the path with the lowest fees;
  - The probing is done by the Lightning node or wallet (hence, it is *not directly seen* by the user).

Path Finding

LN implementations use a very simple path-finding mechanism:

1. Create a **channel graph** from announcements and updates containing the capacity of each channel;
2. Filter the graph, ignoring any channels with insufficient capacity for the amount we want to send;
   - Channels capacity can only be estimated (probabilistic approach);
3. Find paths connecting the sender to the recipient;
4. Order the paths by some weight;
5. Trial-and-error loop: Try each path in order until payment succeeds;
6. Optionally use the HTLC failure returns to update the channel graph, reducing uncertainty;

Path Finding: Definition of Best Path

- Different criteria for defining the *best* path:
  - e.g., with enough liquidity, with low fees, with short timelocks, compliant to specific policies.
- A payment channel is characterized by:
  - **Capacity**: the aggregate amount of satoshis funded with the funding transaction (max amount of value held in the channel). Announced by the gossip protocol.
  - **Balance**: amount of satoshis held by each channel partner that can be sent to the other channel partner;
  - **Liquidity**: portion of the balance that can actually be sent across the channel in one direction (i.e., balance minus reserve and pending HTLCs);
- Uncertainty of balances: We can use failed HTLCs returned from our payment attempts to update our liquidity estimate and reduce uncertainty.

Path Finding and Payment Delivery

- Well-known problem: computing the shortest path!
  - **Fee** and **timelock** information are important for successfully routing the payment;
  - The process of calculating fees happens from the recipient to the sender backward;
  - Dijkstra or A* can be used to search for a path, using fees, estimated liquidity, and timelock delta as a cost function for each hop.
- The sender's node starts the trial-and-error loop by constructing the HTLCs, building the onion, and attempting delivery of the payment.
  - A successful result;
  - An error: the payment can be retried via a different path by updating the graph;
  - No response: no retry possible to avoid a double payment.

Onion Routing

- LN uses an onion routing protocol (SPHINX Mix Format);
- An intermediary node can **only** see on which channel it received an onion and on which channel to forward the onion.
  - Onions can have up to around 26 hops;
  - The onions are small enough to fit into a single TCP/IP packet;
  - The onions are constructed such that they will always have the same length independent of the position of the processing node along the path.
- Onions have an HMAC at each layer so that manipulations of onions are prevented;
  - Onion encrypted using different ephemeral encryption keys for every hop;
- Errors can be sent back to the original sender (using onion routing).

Onion Routing

	IP Routing	Onion Routing
Type	Best effort	Source based
Data format	Open headers	Encrypted headers
Sender / Recipient	Known to routing nodes	Unknow to routing nodes
Address Space	logical-hierarchical overlay network	fully decentralized p2p network
Edge weights	Mostly static (bandwidth)	Highly dynamic (fees, balances)
Topology info propagation	Network can react to change (e.g., BGP)	Gossip is slow / noisy to propagate all relevant information
DoS Attacks	Via spoofing can be mitigated by ISPs	Anyone can send/delay onions. Impossible to mitigate!
Path finding	Collaboratively by the network	Achieved by sender or 3rd party service

Scaling Blockchains

Lightning Network: Concluding Remarks

Main Differences between LN and Bitcoin

- In LN, the recipient of a payment creates an invoice (address is not enough):
  - An invoice is a payment instruction with a payment hash (hash of a random number), a recipient, an amount, and an optional text description;
  - An invoice can **only be used once**;
- A payment results in a **channel balance update** (no UTXOs consumption);
  - Portions of balance can be sent back and forth within the channel;
  - Payments are immediate and almost completely private;
  - A user can only send as much bitcoin as currently exists on his side of a channel;
- The payment recipient needs to be **online** (synchronous payment).
- Blockchain **confirmations** only matter for opening and closing channels;
- Users pay **fees** for routing payments through channels: a minimum base fee plus a fee rate proportional to the payment value;

Scaling Blockchains

Roll-ups

What is a Rollup?

- Idea:
  - Execute transactions off-chain
  - Publish on-chain (L1): compressed data, proof of correctness
- L1 guarantees: security and data availability
- L2 manages: exeuction and scalability

Rollup Archicture

- Key components:
  - Sequencer (orders transactions)
  - Prover/Validator
  - Smart contract on L1
- Pipeline (high-level):
  1. User creates the L2 transaction
  2. The sequencer collects transactions and creates a batch
  3. The batch is then published on L1
  4. Validation through proof (either validity proof or fraud proof)

Optimistic Rollups

- Key assumption: transactions are valid by default.
- Protocols include a fraud proof
  - The sequencer can quickly submit the batch
  - A challenge windows start (e.g., 7 days), to accept fraud proofs and revert the batch
- Pro:
  - Usually simple protocols
  - Good compatibility with L1 ecosystem (e.g., EVM)
- Con:
  - Slow finality
- Examples: Arbitrum, Optimism

Zero-Knowledge Rollups (ZK Rollups)

- Key assumption: transactions should be validated.
- Protocols include a validity proof
  - Expressed using ZK proof
  - The batch include the ZK proof of validity
- Pro:
  - Fast finality
  - Improved security (safety)
- Con:
  - Increased computational cost
  - Expensive prover
- Examples: Polygon zkEVM, ZKsync, Starknet
- Read more: `https://ethereum.org/developers/docs/scaling/zk-rollups`

ZK Proofs in Rollups: High-level Picture (1)

Problem: L1 cannot re-execute all transactions (too expensive)

Idea:

- Execute transactions off-chain
- Prove correctness with a succinct cryptographic proof

Workflow:

1. Transactions T applied to state S
2. New state S' computed off-chain
3. Prover generates proof π
4. L1 verifies π instead of recomputing

ZK Proofs in Rollups: High-level Picture (2)

Key shift:

- From: *re-executing all transactions*
- To: *verifying a proof of correct execution*

Goal:

$$\text{Verify}(\pi, S, S') = \text{true}$$

ZK Proofs for Rollup State Transitions (1)

Goal: Prove correctness of a state transition without re-executing transactions on-chain

Statement:

$$S' = \mathsf{Exec}(S, T)$$

- Public input: $x = (S, S')$, i.e., states S and S'
- Witness: $w = (T, \mathsf{aux})$

ZK Proofs for Rollup State Transitions (2)

Constraint system:

- Encode the computation as an arithmetic circuit / R1CS instance
- Let z denote public inputs, witness, and intermediate values

$$\langle a_i, z \rangle \cdot \langle b_i, z \rangle = \langle c_i, z \rangle$$

- Constraints capture:
  - signature verification
  - balance updates
  - state transition rules

ZK Proofs for Rollup State Transitions (3)

Proof system:

- pp: public parameters of the zkSNARK
- π: proof that the witness satisfies the constraints for input x

$$\pi \leftarrow \mathsf{Prove}(\mathsf{pp}, x, w) \qquad \mathsf{Verify}(\mathsf{pp}, x, \pi) = 1$$

From Constraints to Succinct Proofs (1)

Step 1: Computation → Constraints

- Express execution as constraints (arithmetization, e.g., R1CS, PLONK, AIR)
- Each constraint checks a simple relation:

$$\langle a_i, z \rangle \cdot \langle b_i, z \rangle = \langle c_i, z \rangle$$

Step 2: Constraints → Polynomials

- Constraints are encoded as polynomials
- Valid execution if all constraints are satisfied
- Intuition: checking many constraints becomes checking one algebraic relation

From Constraints to Succinct Proofs (2)

Step 3: Succinct proof

- Prover compresses the computation into a proof π
- Verifier checks π in time independent of the number of transactions $|T|$

$$\text{Verify}(\text{pp}, x, \pi) = 1$$

Key intuition:

- From many checks $\rightarrow$ to one compact proof
- Verify computation *without re-executing it*

Modern Proving Systems for Rollups (1)

Groth16 (Pairing-based SNARK, R1CS):

- Pros: Very small proofs (tens of bytes); Extremely fast on-chain verification;
- Cons: Circuit-specific *trusted setup*; Limited flexibility for evolving circuits.

PLONK (Universal SNARK, Plonkish constraints):

- Pros: Universal and updatable setup; More flexible (good for zkVM / zkEVM);
- Cons: Larger proofs or higher prover cost than Groth16; More complex proving pipeline.

Modern Proving Systems for Rollups (2)

STARK (Transparent, AIR):

- Pros: No trusted setup; Highly scalable proving (large computations);
- Cons: Larger proof sizes; Higher on-chain verification cost.

Key insight:

- Different systems use different *arithmetizations*
- Same goal: encode computation as algebra and prove it succinctly
- Trade-offs between proof size, prover cost, setup, and scalability.

Optimistic vs ZK Rollups

Feature	Optimistic Rollups	ZK Rollups
Validation Model	Fraud proofs (challenge-based)	Validity proofs (zkSNARK/zkSTARK)
Finality	Delayed (challenge period)	Fast (after proof verification)
Security Assumption	Honest challenger exists	Cryptographic soundness
On-chain Cost	Lower upfront, delayed settlement	Higher proving cost, cheap verification
Prover Complexity	Low	High
Proof Size	None	Small (SNARK) / Larger (STARK)
EVM Compatibility	High (near-native)	Increasing (zkEVM evolving)
User Experience	Worse (withdrawal delay)	Better (fast withdrawals)

Key Insight:

- Optimistic Rollups rely on *dispute resolution*
- ZK Rollups rely on *cryptographic proof of correctness*

Comparison with Lightning Network

Aspect	Lightning Network	Rollups
Type	Payment channels	General-purpose
Execution Model	Off-chain state updates	Off-chain execution + on-chain data
Use Case	Payments (micropayments)	Smart contracts and DeFi
Finality	Instant (within channel)	Fast (ZK) / Delayed (Optimistic)
Privacy	High (off-chain, onion routing)	Moderate (data often on-chain)
Data Availability	Not required on-chain	Required on L1
Interactivity	Interactive protocol (channels)	Mostly non-interactive (batched)
Liquidity Constraints	Yes (channel capacity)	No (global state)
On-chain Footprint	Open/close channels only	Compressed transaction batches

- Lightning Network optimizes *payments* via off-chain state channels
- Rollups optimize *computation* via off-chain execution with on-chain guarantees

Scaling Blockchains

Modular Blockchain

Data Availability Problem

- Issue: even though execution is off-chain, data has to be available
  - The verifier/validator needs data
- Solution:
  - Data stored on-chain (e.g., Ethereum's `calldata`);
  - Data Availability Layers
- This is one of the today's bottleneck

Modular Blockcahin

- Separation of responsibility
  - Execution (through Rollups)
  - Consensus (on-chain, L1)
  - Data Availability (dedicated layer)
- Example of solutions: Celestia, eigenDA

Ethereum and its Rollup-centric Roadmap

- Ethereum roadmap is roll-up centric
- https://ethereum.org/roadmap/scaling/
  - L1: security and data availability
  - L2: execution
- Key technologies
  - idea of temporary blobs for rollup-data
  - proto-danksharding: upgrade that reduces rollup fees by introducing temporary data blobs (instead of using CALLDATA); blobs automatically deleted after 18 days.

Scaling Blockchains

Conclusion

Open Challenges

- **Sequencer centralization:** Many rollups rely on a single sequencer, creating trust and censorship risks
- **MEV (Maximal Extractable Value):** Transaction ordering can be exploited for profit, impacting fairness and users
- **ZK prover costs:** Generating proofs is computationally expensive and resource-intensive
- **Cross-rollup interoperability:** Limited seamless communication and asset transfer across rollups
- **User experience:** Bridging delays, withdrawal latency, and complex workflows

Takeaway

- No single solution for scalability
- Several approaches so far, under development: channels, rollups, sharding
- Dominant trend: Layer 2 + modularity
- Direction: scalability off-chain without compromising L1 security.

References (1)

- Ankit Gangwal et al., *A survey of layer-two blockchain protocols*, Journal of Network and Computer Applications **209** (2023), 103539.
- Patrick McCorry et al., *Towards bitcoin payment networks*, Information Security and Privacy (Joseph K. Liu and Ron Steinfeld, eds.), Springer, 2016, pp. 57–76.
- Andrew Miller et al., *Sprites and state channels: Payment networks that go faster than lightning*, Financial Cryptography and Data Security (Ian Goldberg and Tyler Moore, eds.), Springer, 2019, pp. 508–526.
- Joseph Poon and Thaddeus Dryja, *The bitcoin lightning network: Scalable off-chain instant payments*, 2016.

References (2)

Zhimei Sui et al., *MoNet: A fast payment channel network for scriptless cryptocurrency monero*, Proc. of ICDCS'22, 2022, pp. 280–290.

Matteo Nardelli