Scaling the blockchain part II: Rollups

Dan Boneh
Scaling the blockchain: the problem

Transaction rates (Tx/sec):

- Bitcoin: can process about 7 (Tx/sec)
- Ethereum: can process about 15 (Tx/sec)
- The visa network: can process up to 24,000 (Tx/sec)

Can we scale blockchains to visa speeds? … with low Tx fees
How to process more Tx per second

Many ideas:

• Use a faster consensus protocol

• Parallelize: split the chain into independent shards

• Today: Rollups, move the work somewhere else

• Payment channels: reduce the need to touch the chain
  • Requires locking up funds; mostly designed for payments.
Recall: a basic layer-1 blockchain

Can handle 15 Tx/sec ...

World state: balances, storage, etc.

A layer-1 blockchain (e.g., Ethereum)

current world state
updated world state
updated world state
Rollup idea 1: batch many Tx into one

A layer-1 blockchain (e.g., Ethereum)

- Current world state (Rollup state Merkle root)
- Updated world state (updated Rollup state root)
- Updated Rollup state root, and Tx list

Rollup coordinator

- $T_{X_A}$
- $T_{X_B}$
- $T_{X_C}$

Rollup state:
- Alice’s balance
- Bob’s balance
- ...
Rollup idea 1: batch many Tx into one

Key point:

• *Hundreds* of transactions on Rollup state are batched into a *single* transaction on layer-1
  ⇒ 100x speed up in Tx/sec

• Let’s see how ...

A layer-1 blockchain (e.g., Ethereum)

- current world state (Rollup state Merkle root)
- updated world state (updated Rollup state root)

Rollup state:
  Alice’s balance
  Bob’s balance
  ...

(Tx list)
Rollup operation (simplified)

Layer 1 blockchain (e.g. Ethereum)

[A→B: 2 ETH], \( sig_A \)

atomic swap:

[B→Z: 1 ETH]
[Z→B: 2 USDC]
\( sig_B \) \( sig_Z \)

Rollup state root

Merkle Tree

Alice: 5 DAI 3 ETH
Bob: 2 ETH ...
Zoe: 1 ETH 3 USDC

Tx

block 354
Rollup operation (simplified)

Alice:
5 DAI
1 ETH

Bob:
3 ETH
2 USDC

Zoe:
2 ETH
1 USDC

Layer 1 blockchain (e.g. Ethereum)

block 354
block 361

Merkle Tree

new Rollup root

[A→B: 2 ETH], \( s_i g_A \)
[B→Z: 1 ETH]
[Z→B: 2 USDC] \( s_i g_B \ s_i g_Z \)

atomic swap:

Tx

new root, Tx list
In more detail

Rollup contract on layer-1 holds assets of all Rollup accounts (and Merkle state root)

<table>
<thead>
<tr>
<th>Alice: state</th>
<th>Bob: state</th>
<th>Uniswap: state</th>
<th>Rollup contract: 7 ETH, 3 DAI, root</th>
<th>Rollup state (L2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 ETH, 1 DAI</td>
<td>3 ETH, 2 DAI</td>
<td>...</td>
<td>(coordinator stores state)</td>
<td>...</td>
</tr>
</tbody>
</table>

Layer-1 blockchain (L1)
Transfers inside Rollup are easy (L2 $\rightarrow$ L2)

[Alice $\rightarrow$ Bob: 2 ETH, $\sigma_a$]

(with hundreds of Tx)

Rollup state (L2)

Alice: 4 ETH, 1 DAI
Bob: 3 ETH, 2 DAI

Uniswap: state

Rollup contract:
7 ETH, 3 DAI, root

Layer-1 blockchain (L1)
Transfers inside Rollup are easy (L2 → L2)

Coordinator updates root on Rollup contract

Alice: 2 ETH, 1 DAI
Bob: 5 ETH, 2 DAI

Uniswap:

Rollup contract:
7 ETH, 3 DAI, root

Layer-1 blockchain (L1)
Transferring funds into Rollup (L1 → L2)

Alice issues an L1 Tx: slow and expensive

Rollup state (L2)

Alice: 2 ETH, 1 DAI
Bob: 5 ETH, 2 DAI

Alice: state
Bob: state
Uniswap: state
Rollup contract: 7 ETH, 3 DAI, root

Layer-1 blockchain (L1)

2 ETH
Transferring funds into Rollup (L1 → L2)

Alice issues an L1 Tx: slow and expensive

Rollup state (L2)

Alice: 4 ETH, 1 DAI
Bob: 5 ETH, 2 DAI

new Merkle root, Tx list

Alice: state
Bob: state
Uniswap: state
Rollup contract: 9 ETH, 3 DAI, root

Layer-1 blockchain (L1)

[2 ETH], sigA

2 ETH
Transferring funds out of Rollup (L2 → L1)

Requires extra gas on L1 to process transfer

[withdraw 4 ETH], $\text{sig}_A$

(plus hundreds of Tx)

Rollup state

<table>
<thead>
<tr>
<th>Alice:</th>
<th>Bob:</th>
<th>Rollup contract:</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 ETH, 1 DAI</td>
<td>5 ETH, 2 DAI</td>
<td>9 ETH, 3 DAI, root</td>
</tr>
</tbody>
</table>

Layer-1 blockchain (L1)
Transferring funds out of Rollup (L2 → L1)

Requires extra gas on L1 to process transfer

[w] [withdraw 4 ETH], $\sigma_A$

(plus hundreds of Tx)

Alice: 0 ETH, 1 DAI
Bob: 5 ETH, 2 DAI

Uniswap: state
Rollup contract: 5 ETH, 3 DAI, root

New Merkle root, Tx list

Layer-1 blockchain (L1)
Transactions within a Rollup are easy:
• Batch settlement on L1 network (e.g., Ethereum)

Moving funds into or out of Rollup system (L1 ↔ L2) is expensive:
• Requires posting more data on L1 network ⟹ higher Tx fees.

Moving funds from one Rollup system to another (L2 ↔ L2)
• Either via L1 network (expensive),
  or via a direct L2 ↔ L2 bridge (cheap)
Two copies of Uniswap

<table>
<thead>
<tr>
<th>Alice: state</th>
<th>Bob: state</th>
<th>Uniswap: state</th>
<th>Rollup contract:</th>
<th>Layer-1 blockchain (L1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 ETH, 1 DAI</td>
<td>3 ETH, 2 DAI</td>
<td>7 ETH, 3 DAI, root</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Rollup state (L2)

⇒ Rollup users can cheaply interact with Uniswap on Rollup
Running contracts on a Rollup?

Coordinator maintains state of all contracts on Rollup system:

- It updates the Uniswap Merkle leaf after every Tx to Uniswap
- Writes updated Rollup state Merkle root to L1 chain
Running contracts on a Rollup?

<table>
<thead>
<tr>
<th>Rollup state (L2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniswap: state</td>
</tr>
<tr>
<td>Alice: 4 ETH, 1 DAI</td>
</tr>
<tr>
<td>Bob: 3 ETH, 2 DAI</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

Rollup functions as Ethereum, but w/o a consensus protocol!!

- It relies on the L1 chain to attest to the current Rollup state
Enduser configures its wallet to send Tx to the RPC points of the selected Rollup.

(by default Metamask sends Tx to the Ethereum Mainnet RPC points)
Not so simple ...
Problems ...

Problem 1: what if coordinator is dishonest?
• It could steal funds from the Rollup contract
• It could issue fake Tx on behalf of users

Problem 2: what if coordinator stops providing service?
• If Rollup state is lost, how can we initialize a new coordinator?
Problem 1: what if coordinator is dishonest?

Can coordinator steal funds from Rollup users?

No! L1 chain verifies that Rollup state updates are valid.

⇒ all Tx are valid and properly signed by the Rollup users

Challenge: how to do this cheaply ?? (with little gas on L1)

<table>
<thead>
<tr>
<th>Alice: state</th>
<th>Bob: state</th>
<th>Rollup contract: 7 ETH, 3 DAI, root</th>
<th>...</th>
</tr>
</thead>
</table>

Layer-1 blockchain (L1)
Verifying Rollup state updates

Approach 1: validity proofs (called a zk-Rollup)

Succinct proof proves that a batch of hundreds of Tx is valid

Layer 1 blockchain (e.g. Ethereum)

Rollup contract accepts new root only if valid proof

Rollup coordinator

updated state root

SNARK proof of valid Tx

Tx list
What the SNARK proof proves

SNARK proof is **short** and **fast** to verify:

⇒ Cheap to verify proof on the slow L1 chain (with EVM support) (usually not a zero knowledge proof)

**Public statement**: (old state root, new state root, Tx list)

**Witness**: (state of each touched account pre- and post- batch, Merkle proofs for touched accounts, user sigs)

SNARK proof proves that:

(1) all user sigs on Tx are valid,  (2) all Merkle proofs are valid,  
(3) post-state is the result of applying Tx list to pre-state
zkEVM

When a contract (e.g. Uniswap) runs on a Rollup:

• coordinator builds a SNARK proof of correct execution of an EVM program ⇒ called a zkEVM

• Generating proof is a heavyweight computation… verifying proof is fast

Two flavors of zkEVM:
• Prove that EVM bytecode ran correctly (Polygon zkEVM, Scroll)
• Compile Solidity to a SNARK-friendly circuit (MatterLabs)
The end result

Rollup contract on L1 ensures coordinator cannot cheat:

• all submitted Tx must have been properly signed by users
• all state updates are valid

⇒ Rollup contract on L1 will accept any update with a valid proof
⇒ Anyone can act as a coordinator (with enough compute power)
Approach 2: **fraud proofs** (called an **optimistic Rollup**)
- Coordinator deposits stake in escrow on L1 Rollup contract
- Operation: Coordinator submits state updates to L1 w/o a proof
  - If update is invalid: anyone has seven days to submit a fraud proof
    - Successful fraud proof means coordinator gets *slashed* on L1
    - Unsuccessful fraud proof costs complainer a fee

**Challenge:** how to prove fraud to Rollup contract on L1 ??

Naively: L1 can re-execute all Tx in batch ⇒ expensive and slow
Fraud Proof game

Coordinator computes Merkle tree of all states.
Sends Merkle root to L1

pre-root
Tx list

coordinator
claimed post-root

fraud claim
different post-root

pre-state
break computation into small steps
post-state
Fraud Proof game

pre-root

Tx list

we know $state_n \neq state'_n$

fraud claim

different $state'_n$

coordinator

claimed $state_n$

Merkle root

hash$_{[0 \rightarrow n/4]}$

hash$_{[n/4 \rightarrow n/2]}$

hash$_{[n/2 \rightarrow n]}$

hash$_{[0 \rightarrow n/2]}$

$state_0$

$state_n/2$

$state_n$
Fraud Proof game: binary search

Suppose $state_{n/2} \neq state'_{n/2}$

Merkle root

hash$_{[0 \rightarrow n/2]}$, hash$_{[n/2 \rightarrow n]}$, hash$_{[0 \rightarrow n/4]}$, hash$_{[n/4 \rightarrow n/2]}$
Fraud Proof game: binary search

Suppose $state_{n/2} \neq state'_{n/2}$

Coordinator sends $hash_{[0 \rightarrow n/2]}$ to L1
Alice sends “left” to L1
Suppose $state_{n/4} = state'_{n/4}$

Coordinator sends hash$_{[n/4 \rightarrow n/2]}$ to L1

Alice sends “right” to L1
Fraud Proof game: binary search

Suppose $state_{n/4} = state'_{n/4}$

Coordinator sends $hash_{[n/4 \rightarrow n/2]}$ to L1

Alice sends “right” to L1
Fraud Proof game: binary search

After $\log_2 n$ rounds:
- $L_1$ has $state_i$ and $state_{i+1}$ from coordinator
- $state_i = state_i'$ and $state_{i+1} \neq state_{i+1}'$

or game times out because one player defects

⇒ Now $L_1$ can verify fraud proof by checking one computation step!
Some difficulties

(1) Transactions only settle after 7 days (after fraud window expires)
• Alice needs to wait 7 days to withdraw funds from Rollup (Rollup contract will only send her the funds after 7 days)

For fungible tokens, a 3\textsuperscript{rd} party can advance the funds to Alice after checking validity of Alice’s withdraw Tx. Does not apply to non-fungible tokens.

(2) Suppose a successful fraud proof 4 days after batch is posted
⇒ all subsequent Tx need to be resubmitted
The end result

Can easily port any smart contract to an optimistic Rollup
• The Rollup EVM can be enhanced with new features (opcodes)

High Tx throughput: in principle, up to 4000 tx/s
• No need for special hardware at the coordinator

Anyone can act as a coordinator and a verifier

Downside: 7 day finality delay
An example (StarkNet -- using validity proofs)

<table>
<thead>
<tr>
<th>Number</th>
<th>Hash</th>
<th>Status</th>
<th>Num. of Tx</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>PENDING</td>
<td>PENDING</td>
<td>PENDING</td>
<td>64</td>
<td>3min</td>
</tr>
<tr>
<td>13011</td>
<td>0x0432_2380</td>
<td>ACCEPTED_ON_L2</td>
<td>82</td>
<td>8min</td>
</tr>
<tr>
<td>13010</td>
<td>0x0492_f0d1</td>
<td>ACCEPTED_ON_L2</td>
<td>122</td>
<td>15min</td>
</tr>
<tr>
<td>13009</td>
<td>0x0081_b7af</td>
<td>ACCEPTED_ON_L2</td>
<td>127</td>
<td>24min</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12868</td>
<td>0x060c_15eb</td>
<td>ACCEPTED_ON_L2</td>
<td>58</td>
<td>8h</td>
</tr>
<tr>
<td>12867</td>
<td>0x0654_3b0f</td>
<td>ACCEPTED_ON_L1</td>
<td>72</td>
<td>9h</td>
</tr>
<tr>
<td>12866</td>
<td>0x0779_57d6</td>
<td>ACCEPTED_ON_L1</td>
<td>63</td>
<td>9h</td>
</tr>
<tr>
<td>12865</td>
<td>0x06ae_943f</td>
<td>ACCEPTED_ON_L1</td>
<td>97</td>
<td>9h</td>
</tr>
</tbody>
</table>

Tx posted on L1 (Ethereum) about every eight hours

Source: starkscan.co
An example (Optimism -- using fraud proofs)

<table>
<thead>
<tr>
<th>Txn Batch</th>
<th>Age</th>
<th>Batch Size</th>
<th>L1 Txn Hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>328411</td>
<td>2 mins ago</td>
<td>109</td>
<td>0xbb358889959cf83413…</td>
</tr>
<tr>
<td>328410</td>
<td>2 mins ago</td>
<td>91</td>
<td>0x8398475c9b7179ebfe…</td>
</tr>
<tr>
<td>328409</td>
<td>3 mins ago</td>
<td>85</td>
<td>0x3264a772e220beca85…</td>
</tr>
<tr>
<td>328408</td>
<td>3 mins ago</td>
<td>106</td>
<td>0xa92bd044f7576a87c1…</td>
</tr>
<tr>
<td>328407</td>
<td>4 mins ago</td>
<td>101</td>
<td>0x302cda229ed83d570e…</td>
</tr>
<tr>
<td>328406</td>
<td>4 mins ago</td>
<td>79</td>
<td>0x0f205018c4a289af9d7…</td>
</tr>
<tr>
<td>328405</td>
<td>5 mins ago</td>
<td>113</td>
<td>0xedbe2e0706cb06c3cb…</td>
</tr>
<tr>
<td>328404</td>
<td>5 mins ago</td>
<td>120</td>
<td>0xffaa82d2f006f519a892…</td>
</tr>
</tbody>
</table>

Shows batch posted on L1 (Ethereum)

Source: optimistic.etherscan.io
... ok, so coordinator cannot submit invalid Tx.

Problem 2: What if coordinator stops providing service?

Solution: setup a new coordinator

... but need the latest Rollup state

Where to get state?? The data availability problem
Ensuring Rollup state is always available

The definition of a Rollup:

Rollup state can always be reconstructed from data on the L1 chain

Layer 1 blockchain (e.g. Ethereum)

Rollup contract

state root

Tx list

updated state root

Sent to Rollup contract on L1 as part of state update message
Ensuring Rollup state is always available

To reconstruct current Rollup state:
• Read all Rollup update messages and re-execute Tx.
  ⇒ anyone can become a coordinator
• Rollups use L1 for data storage

What to store?
• For zk-Rollup: send Tx summary to L1, without user signatures
  (SNARK proof proves validity of signatures)
• For optimistic: need to send Tx summary *and* signatures to L1
Ensuring Rollup state is always available

The downside: **expensive**

- Tx list is sent as calldata: 16 gas per non-zero byte
  
  (EIP-4488 aims to support Rollups by reducing to 3 gas/byte)

In practice:

- Optimistic Rollups fee/Tx: 3-8 times lower than Ethereum L1
- zk-Rollup fee/Tx: 40-100 times lower than Ethereum L1

Can we do even better?
To further reduce Tx fees:

- **Store L2 state root** (small) on the L1 chain
- **Store Tx data** (large) with a Data Availability Committee (DAC):
  - comprises a set of nodes trusted to keep the data available
  - cheaper than storage on L1
  - L1 accepts an update only if all DAC members sign it
  ⇒ ensures that all DAC members accepted Tx data

Setting up a new coordinator depends on availability of the DAC
Validium: an L2 using a DAC and validity proofs (SNARKs)

- Well suited for lower value assets.
- Potential privacy benefits ... only DAC members see Tx data

An example: StarkEx uses a five member DAC

- Users can choose between Validium or Rollup modes

  (Tx data off-L1-chain vs. Tx data on-L1-chain)

  cheaper Tx fees, More expensive Tx, but only secure as DAC but same as L1 security
## Summary: types of L2

### Scaling the blockchain:
Payment channels and Rollups (L2 scaling)

<table>
<thead>
<tr>
<th></th>
<th>SNARK validity proofs</th>
<th>Fraud proofs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tx data on L1 chain</td>
<td>zkRollup</td>
<td>optimistic Rollup, 7 day finality</td>
</tr>
<tr>
<td>Tx data in a DAC</td>
<td>Validium (reduced fees, but higher risk)</td>
<td>“Plasma”</td>
</tr>
</tbody>
</table>
## Volume of some L2 systems

<table>
<thead>
<tr>
<th>Tx Volume/day</th>
<th>average fee/tx</th>
<th>(on Nov. 15, 2022)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Ethereum:</td>
<td>1013K Tx</td>
<td>2.71 USD/Tx</td>
</tr>
<tr>
<td>• Arbitrum:</td>
<td>345K Tx</td>
<td>0.08 USD/Tx</td>
</tr>
<tr>
<td>• Optimism:</td>
<td>303K Tx</td>
<td>0.13 USD/Tx</td>
</tr>
<tr>
<td>• StarkNet:</td>
<td>14K Tx</td>
<td>0.22 USD/Tx</td>
</tr>
</tbody>
</table>
Can coordinator censor a Tx?

What if coordinators refuse to process a Tx?

What to do? One option:

• enduser can post Tx directly to the L1 Rollup contract
• The L1 Rollup contract will then refuse to accept updates until an update includes that Tx

⇒ censorship will cause the entire Rollup to freeze
SNARK recursion

Layer 3 and beyond ...
SNARK recursion

Two level recursion: proving knowledge of a proof

Use $V'(vp', x, \pi')$ to verify final proof $\pi'$

public: $x$
witness: $w$

proves $P$ knows $w$ s.t.
$C(x, w) = 0$

proves $P'$ knows $\pi$ s.t.
$V(vp, x, \pi) = yes$

SNARK prover $P$

$\pi$

$\pi'$

$(S, P, V)$

$\pi'$

$(S', P', V')$
**Application 1: proof compression**

public: $x$

witness: $w$

SNARK prover $P$

prove $C(x, w) = 0$

fast prover, but outputs a large proof

$(S, P, V)$

SNARK prover $P'$

prove $V(\nu p, x, \pi) = yes$

$(S', P', V')$

Use $V'(\nu p', x, \pi')$ to verify final short proof $\pi'$
Application 2: Layer three and beyond

Alice: state
Bob: state

L3 Rollup contract: state root
Alice: 2 ETH, 1 DAI
Bob: 5 ETH, 2 DAI

L2 Rollup state

Alice: state
Uniswap: state

Rollup contract: 7 ETH, 3 DAI, root

Layer-1 blockchain (L1)
Layer three and beyond

One L2 coordinator can support many L3s
- each L3 can run a custom VM with its own features
- L3 chains can communicate with each other through L2

Each L3 coordinator submits Tx list and SNARK proof to L2
- L2 coordinator: collects batch of proofs,
  - builds a proof $\pi$ that it has a batch of valid proofs, and
  - submits the single proof $\pi$ and updated root to L1 chain.

$\Rightarrow$ Scaling factor $100 \times 100$
END OF LECTURE

Next lecture: more applications