GRAPH-BASED PROTOCOLS

Kai Mast
CS639/839
Spring 2023

Announcements

- More great upcoming talks
  - Dahlia Malkhi from Chainlink (formerly Diem) on Monday
- Project 2b extended until Monday night
- I will get back to you soon about project proposals!

TODAY'S AGENDA

- Background on concurrent transaction processing
  - Serializability
  - Intra-block transaction ordering

- IOTA: A flawed approach

- Avalanche
  - Snowman

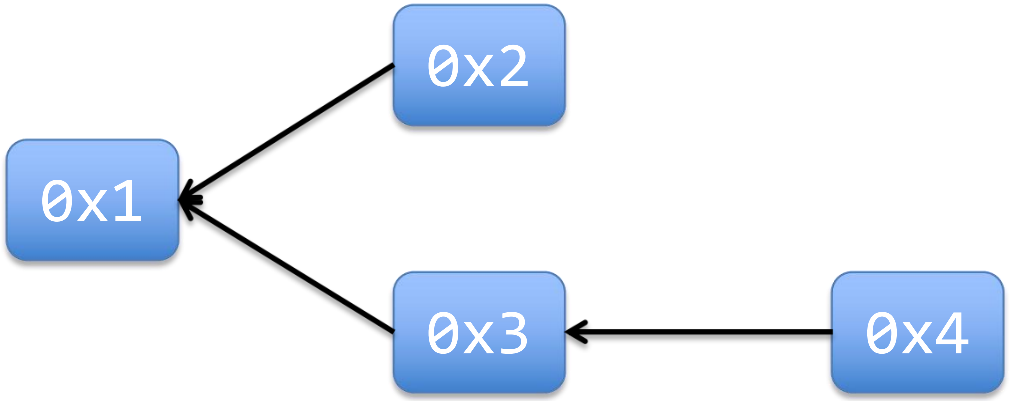
RECAP: PROOF OF STAKE

Ouroboros (v1)
- Randomly pick block creators for each time slot
- Probabilistic block confirmation, like Bitcoin
- Synchronous network model

Algorand
- Randomly pick block creator and committee
- Blocks are confirmed after committee approves a created block
- Partially synchronous network model

Ethereum 2.0
- Ouroboros-like block creation + Casper (Finality Gadget)
- Block are confirmed in two rounds
  - first “justified”, then “finalized”
- Partially synchronous network model

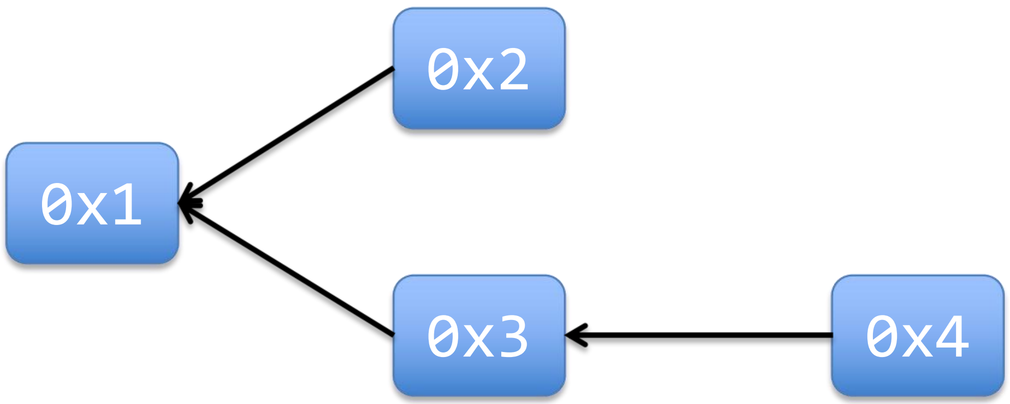
PARTIAL ORDERS W/ UTXOS



Identifier	Inputs	Outputs
0x1	[0xA:0]	[1,1]
0x2	[0x1:1]	[1]
0x3	[0x1:0]	[0.5,0.5]
0x4	[0x3:0, 0x3:1]	[1]

PARTIAL ORDERS W/ UTXOS

Input Set
0xA:0



Output Set
0x2:0
0x4:1

Possible Serial Execution Orders
0x1, 0x2, 0x3, 0x4
0x1, 0x3, 0x4, 0x2
0x1, 0x3, 0x2, 0x3

0x2 can execute independently from 0x3 and 0x4!

SERIALIZABILITY

A common property enforced by database transactions

- Allows parallelism between unrelated operations
- Ensures execution is equal to some serial execution
- The I in ACID (for "Isolation")

Can be enforced through locking or by tracking dependencies (e.g. using UTXOs)

TRANSACTION ORDER IN BITCOIN

Many Bitcoin-like chains follow *Topological Transaction Ordering (TTOR)* in their blocks

- The first transaction is the coinbase transaction (payment to the miner),
- All other transactions must respect the input/output dependencies

TTOR can enable some concurrency when executing/validating a block

DAG-BASED PROTOCOLS

Idea

- Remove notion of blocks entirely
- Network directly agrees on a directed acyclic graph of transactions

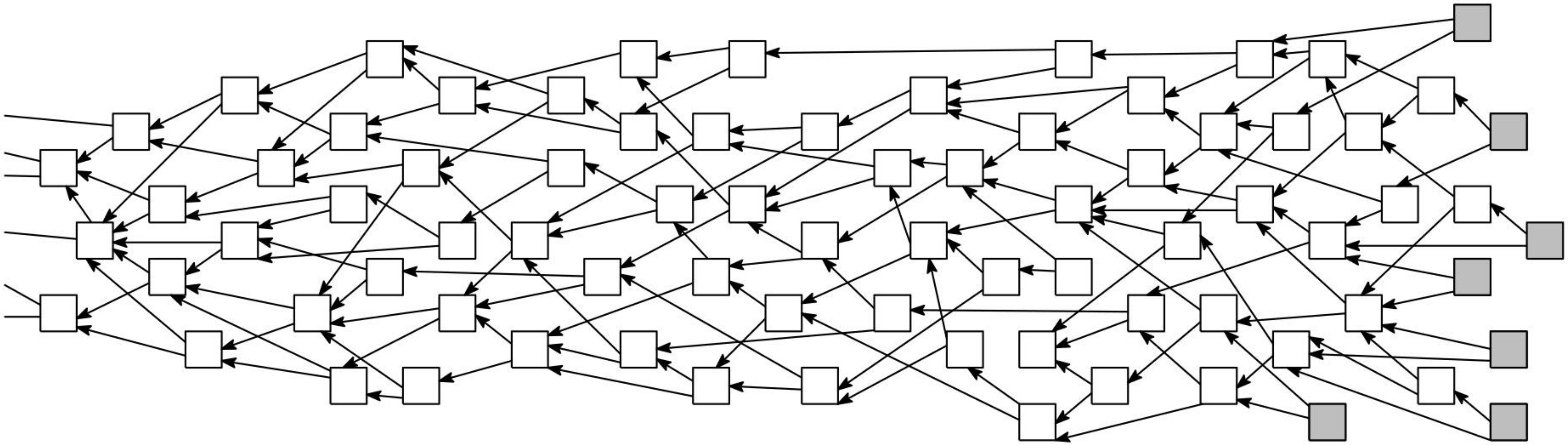
Works well with the UTXO model

- For an honest client, a transaction never conflicts with another
- An attacker might still issue conflicting “rogue” transactions

Problem

- When is a transaction confirmed?

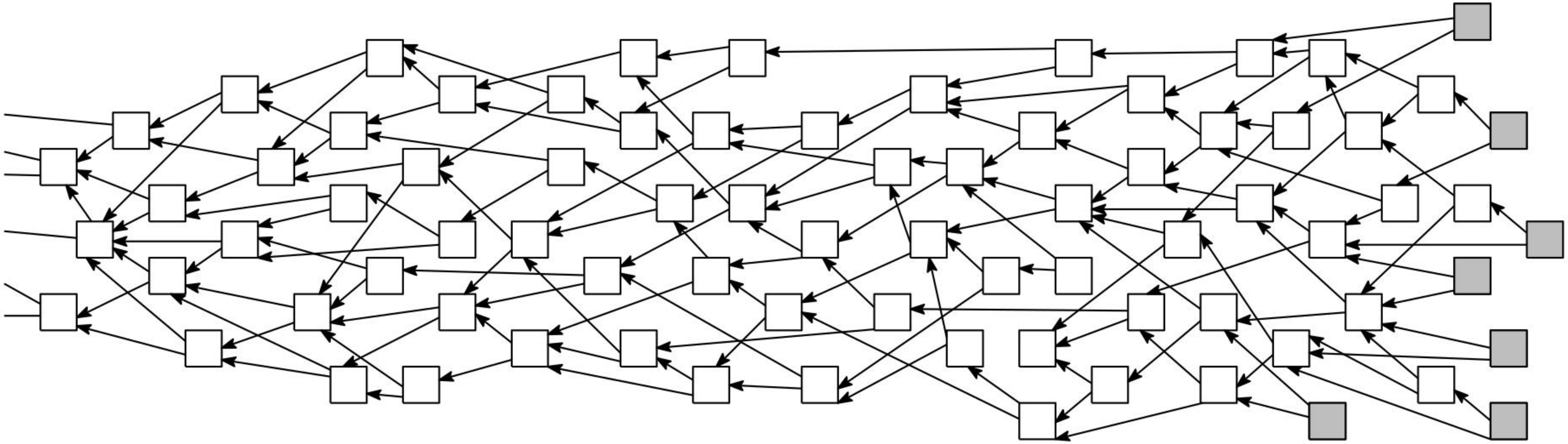
IOTA



Anyone can issue a transaction

- Transactions need to include a proof of work to reduce spam
- Each transaction references at least two predecessors and confirmst them

ATTACKING IOTA



What could go wrong?

- Attacker's confirm their own transaction by creating other transactions
- The DAG could diverge into multiple conflicting branches

IOTA'S BAND-AID: THE COORDINATOR

IOTA relies on a centralized entity to frequently checkpoint the network

- Creates *milestones* (empty transactions) in frequent intervals that confirms all valid transactions

This entity can stop the network from finalizing transactions if it does not issue new milestones

Home / Tech / Security

IOTA cryptocurrency shuts down entire network after wallet hack

Hackers exploit vulnerability in official IOTA wallet to steal millions

Written by **Catalin Cimpanu,** Contributor on Feb. 15, 2020

AVALANCHE

An new type of consensus protocols
- Relies on probabilistic sampling

Similar properties as Nakamoto consensus
- Works with very large networks
- Can handle changing membership
- Constant communication complexity per node and round
  - Larger networks need more communication rounds

Differences to Nakamoto consensus
- Requires knowledge of most nodes in the network
- Can work with, both, DAGs and chains
- Fast confirmation times

SLUSH

Simplified version of Avalanche

- No Byzantine fault tolerance
- Binary decision (either “Red” or “Blue”)

Some nodes start with a specific color

- Others adopt a color when first queried

Each round, nodes sample a random set of `k` other nodes

- If a majority of sampled peers responds with the same color, adopt the color
- Majority is defined by parameter α

Execut for `m` rounds

- Protocol will reach agreement with a high enough value of high `m`
- `m` grows logarithmically with the network size

procedure $\textsc{onQuery}(v, col')$
    if $col = \bot$ **then** $col := col'$
    $\textsc{respond}(v, col)$
procedure $\textsc{slushLoop}(u, col_0 \in \{\mathtt{R}, \mathtt{B}, \bot\})$
    $col := col_0$ // initialize with a color
    for $r \in \{1 \ldots m\}$ **do**
        // if $\bot$, skip until $\textsc{onQuery}$ sets the color
        if $col = \bot$ **then continue**
        // randomly sample from the known nodes
        $\mathcal{K} := \textsc{sample}(\mathcal{N} \backslash u, k)$
        $P := [\textsc{query}(v, col) \quad$ **for** $v \in \mathcal{K}]$
        for $col' \in \{\mathtt{R}, \mathtt{B}\}$ **do**
            if $P.\textsc{count}(col') \geq \alpha$ **then**
                $col := col'$
    $\textsc{accept}(col)$

SNOWFLAKE

Slush with Byzantine Fault-Tolerance

- Binary decision (either “Red” or “Blue”)

Some nodes start with a specific color

- Others adopt a color when first queried

Each round, nodes sample a random set of `k` other nodes

- If a majority of sampled peers respond with the same color, adopt that color
- If we already have adopted the color, increase counter `cnt`
- If we have not adopted the color yet, reset the counter

Run until confidence exceeds some threshold β

procedure $\textsc{snowflakeLoop}(u, col_0 \in \{\mathtt{R}, \mathtt{B}, \bot\})$
    $col := col_0,\ cnt := 0$
    while undecided **do**
        if $col = \bot$ **then continue**
        $\mathcal{K} := \textsc{sample}(\mathcal{N} \backslash u, k)$
        $P := [\textsc{query}(v, col)$ **for** $v \in \mathcal{K}]$
        $maj := \mathtt{false}$
        for $col' \in \{\mathtt{R}, \mathtt{B}\}$ **do**
            if $P.\textsc{count}(col') \geq \alpha$ **then**
                $maj := \mathtt{true}$
                if $col' \neq col$ **then**
                    $col := col',\ cnt := 1$
                else $cnt{+}{+}$
                if $cnt \geq \beta$ **then** $\textsc{accept}(col')$
        if $maj = \mathtt{false}$ **then** $cnt := 0$

SNOWBALL

Like Snowflake, but

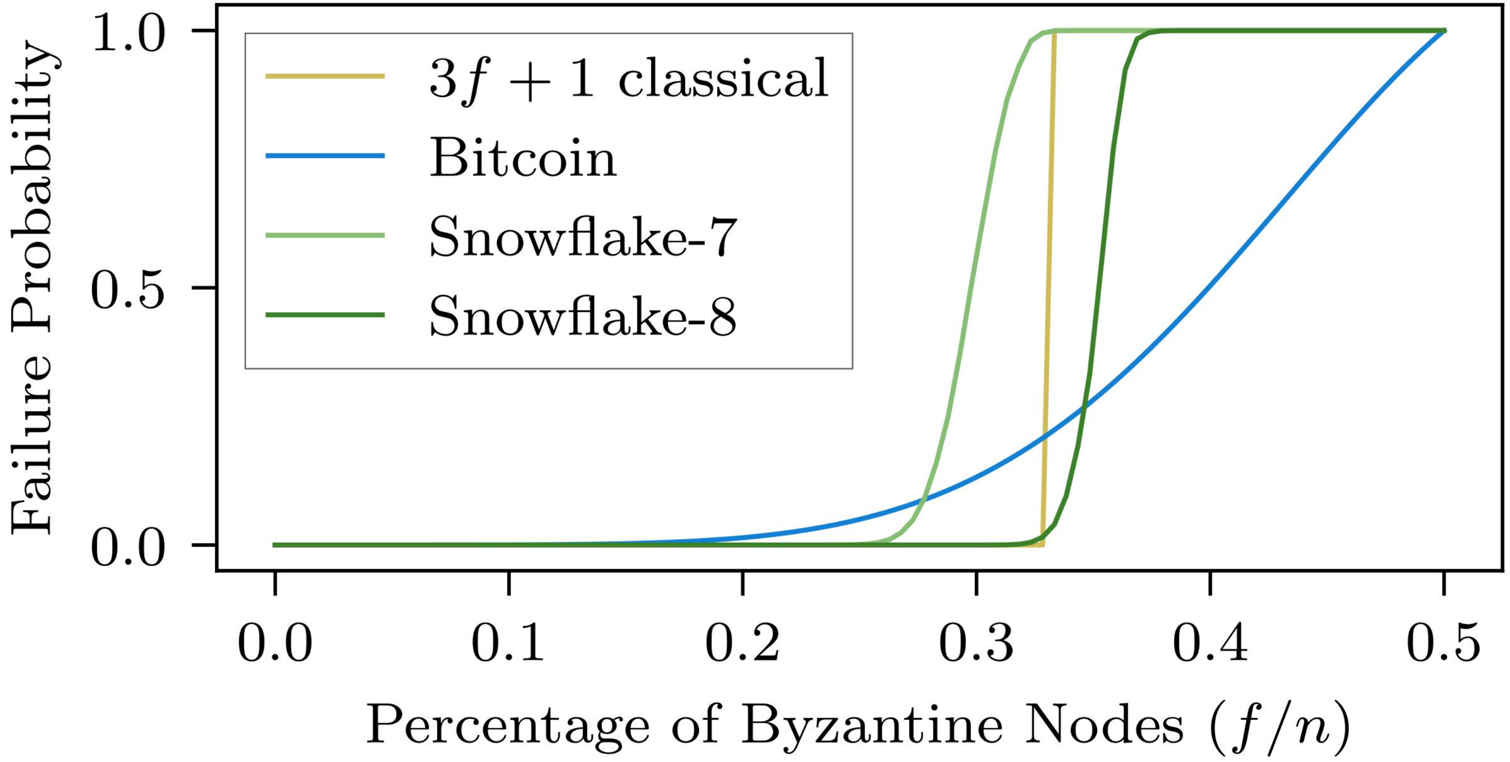
- Keeps track of majorities reached per color as *confidence value* `d`
- Confidence is increased every time a majority is reached in a voting round
- Only change decision when confidence for one color exceeds confidence for the other color

More resistant against Byzantine actors

```
procedure snowballLoop(u, col_0 ∈ {R, B, ⊥})
    col := col_0, lastcol := col_0, cnt := 0
    d[R] := 0, d[B] := 0
    while undecided do
        if col = ⊥ then continue
        K := sample(N \ u, k)
        P := [query(v, col)   for v ∈ K]
        maj := false
        for col' ∈ {R, B} do
            if P.count(col') ≥ α then
                maj := true
                d[col']++
                if d[col'] > d[col] then
                    col := col'
                if col' ≠ lastcol then
                    lastcol := col', cnt := 1
                else cnt++
                if cnt ≥ β then accept(col')
        if maj = false then cnt := 0
```

SNOWBALL SIMULATION

SNOWBALL FAULT TOLERANCE



With k=10 and β=250 for Avalanche

For Bitcoin, it models the probability that a block with 6 confirmations (1 hour) will be reorganized

AVALANCHE

Transactions can reference any number of predecessors

- They do not actually need to depend on the respective UTXOs
- A transaction is only valid if none of its predecessors conflict
- Transactions can be re-issued if one of its parents conflict

Extends snowball with the notion of a DAG

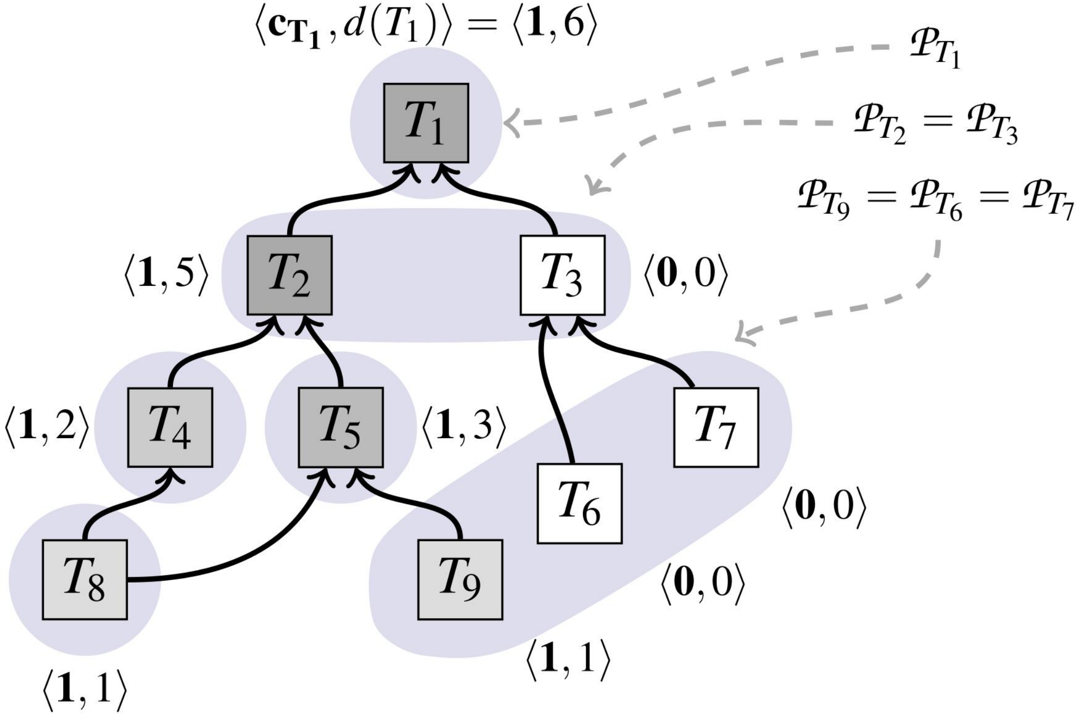
- Confirming a transaction in the DAG, also confirms all its predecessors
- Need not confirm every transaction, but sets of transactions

AVALANCHE

Avalanche queries the network at most once per transaction

- Predecessor of the transaction inherit its count and confidence
- Transactions are eventually confirmed by their predecessors in the absence of conflicts
- No-op transactions can be inserted to allow for additional voting rounds

Avalanche batches queries about multiple transactions if possible

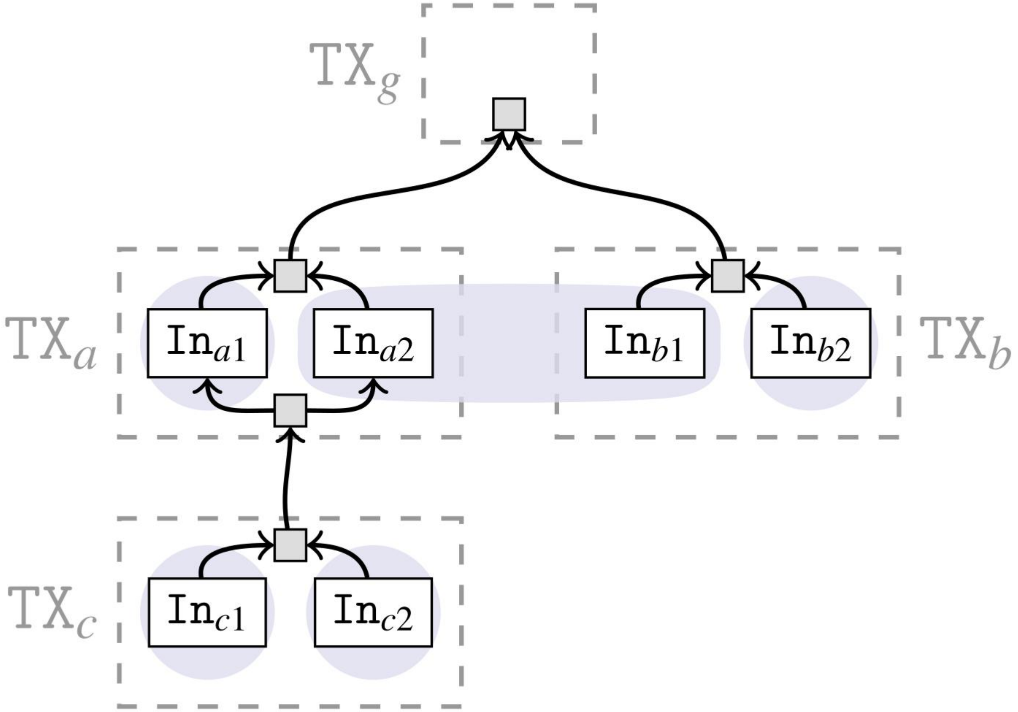


- Shows <counter, confidence> for each transaction
- Shaded areas are conflicting transactions

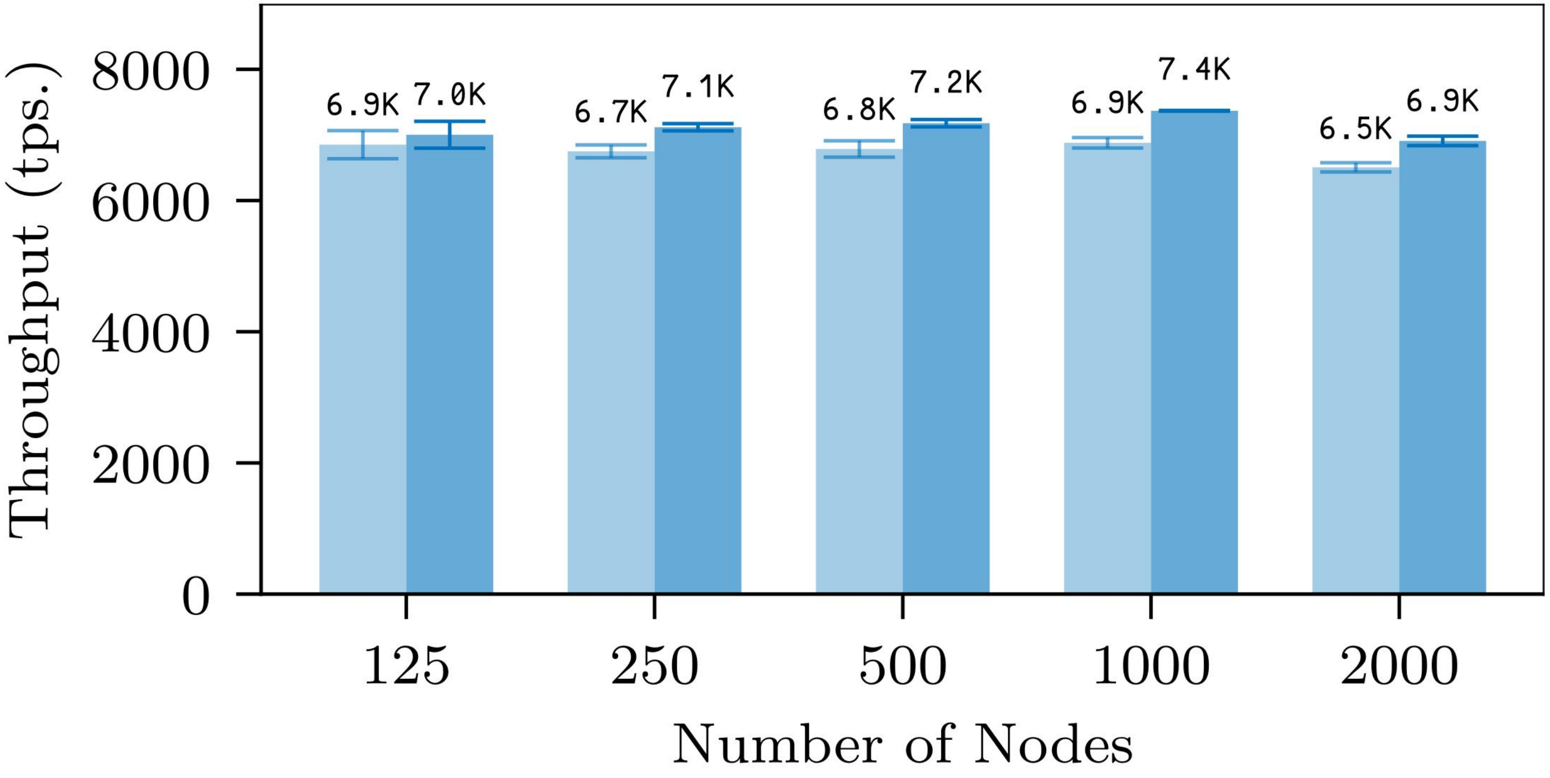
BATCHING IN AVALANCHE

Achieve consensus on every UTXO input, not the transaction as a whole

- Allows for multi-input transactions
- We can batch queries about different UTXOs in a single request for efficiency



AVALANCHE PERFORMANCE



Throughput of Avalanche without geo-replication

Light blue bars show batch size 20 and right bars batch size 40.

AVALANCHE IN THE WILD

Exchange Chain

- Creates and transfers tokens
- Uses a DAG

Platform-Chain

- Keeps track of validators (stakers) and other metadata
- Uses a blockchain

Contract Chain

- Supports EVM smart contracts
- Uses a blockchains

How can we support a blockchain (not a DAG) in Avalanche?

SNOWMAN CONSENSUS

Any validator can propose a block

- New blocks are created every ~2 seconds

Use Avalanche to decide between blocks

- Compare all block hashes
- Decide on conflicting bits in the block hashes
- Multiple rounds of binary consensus

Snowman++

- Only allow a random subset of the validators to create blocks
- Reduces conflicts

THAT'S ALL FOR TODAY

Next time:

- Some more info on Avalanche subnetworks
- A deep dive into blockchain node storage