Avalanche: metastable gossip consensus

Péter Garamvölgyi

outline

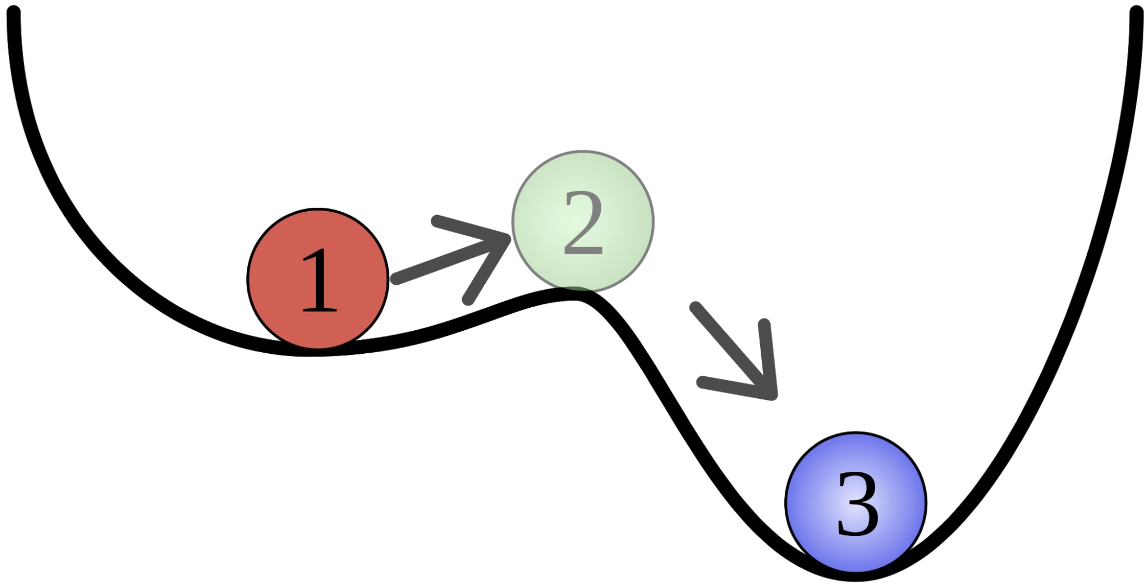
- **what is metastability?**
- the Snow consensus family
  - Slush
  - Snowflake
  - Snowball
- Avalanche

what is metastability?

*“Metastable state is a concept in physics [...] particles may be in a **mixed state**, but if properly induced, they can **quickly stabilize**, that is, all particles share the same state [...] In this state, the particles phase shift to form an invariable and stable structure. It's a dynamic chaotic process that stabilizes steadily, which is exactly what we want to happen in distributed systems.”* *

* E. G. Sirer in an interview https://www.good-blockchain.com/article?id=794

what is metastability?



1: metastable state

2: unstable state

3: stable state

https://en.wikipedia.org/wiki/Metastability

metastability in binary consensus

- nodes need to decide between **RED** and **BLUE**
  - → 50%–50%: unstable (bivalent) state
  - → 51%–49%: metastable state
  - → 99%– 1%: stable state
- goal: quick and robust convergence to an irreversible stable state

outline

- what is metastability?
- the Snow consensus family
  - **Slush**
  - Snowflake
  - Snowball
- Avalanche

Slush: non-BFT metastable binary consensus

```
def onQuery(v, col):
  if color == None: color = col
  respond(v, color)

def slush(me, col0):
  color = col0

  for _ in range(m):
    if color == None: return None

    K = sample(N \ {me}, k)
    P = map(lambda v: query(v, color), K)

    for c in {RED, BLUE}:
      if P.count(c) >= alpha: color = c

  return color
```

main idea:

iterative random sampling

parameters:

N:	full (?) view of network participants
m:	number of rounds
k:	sample size
alpha:	majority parameter (alpha > k/2)

Scalable and Probabilistic Leaderless BFT Consensus through Metastability. (2019).

Slush: non-BFT metastable binary consensus

```
def onQuery(v, col):
  if color == None: color = col
  respond(v, color)

def slush(me, col0):
  color = col0

  for _ in range(m):
    if color == None: return None

    K = sample(N \ {me}, k)
    P = map(lambda v: query(v, color), K)

    for c in {RED, BLUE}:
      if P.count(c) >= alpha: color = c

  return color
```

properties

almost memoryless; no history

O(nk) communication overhead (k << n)

random sampling breaks 50/50 ties

Slush: irreversibility

```
def onQuery(v, col):
  if color == None: color = col
  respond(v, color)

def slush(me, col0):
  color = col0

  for _ in range(m):
    if color == None: return None

    K = sample(N \ {me}, k)
    P = map(lambda v: query(v, color), K)

    for c in {RED, BLUE}:
      if P.count(c) >= alpha: color = c

  return color
```

model using Continuous-Time Markow Chains

state at time t: $\mathcal{S}_t = n/2 + \delta$

i.e. $\boldsymbol{\delta}$ more **BLUE** than **RED**

prob of reverting to minority value bounded by

$$\xi_\delta \leq \left(\frac{1/2 - \delta/n}{\alpha/k}\right)^{\alpha} \left(\frac{1/2 + \delta/n}{1 - \alpha/k}\right)^{k-\alpha}$$

$$\leq e^{-2((\alpha/k)-(1/2)+(\delta/n))^2 k}$$

... drops exponentially with $\boldsymbol{\delta}$

... can be arbitrarily small by tuning $\boldsymbol{k}$ and $\boldsymbol{\alpha}$

Slush: safety

```
def onQuery(v, col):
  if color == None: color = col
  respond(v, color)

def slush(me, col0):
  color = col0

  for _ in range(m):
    if color == None: return None

    K = sample(N \ {me}, k)
    P = map(lambda v: query(v, color), K)

    for c in {RED, BLUE}:
      if P.count(c) >= alpha: color = c

  return color
```

what about Byzantine nodes?

when nodes develop preference for one color

… adversaries can flip nodes to the opposite

… keeping the network in balance

… and preventing consensus.

outline

- what is metastability?
- the Snow consensus family
  - Slush
  - **Snowflake**
  - Snowball
- Avalanche

Snowflake: adding BFT

```
def snowflake(me, col0):
  color = col0
  count = 0

  while True:
    if color == None: return None
    K = sample(N \ {me}, k)
    P = map(lambda v: query(v, color), K)

    for c in {RED, BLUE}:
      if P.count(c) >= alpha:
        count = 0 if color != c else count+1
        color = c

    if count > beta: return color
```

new idea:

nodes must explicitly detect irreversibility

capture strength of conviction using a counter

decide after *beta* identical consecutive samples

new parameters:

beta: decision threshold

Scalable and Probabilistic Leaderless BFT Consensus through Metastability. (2019).

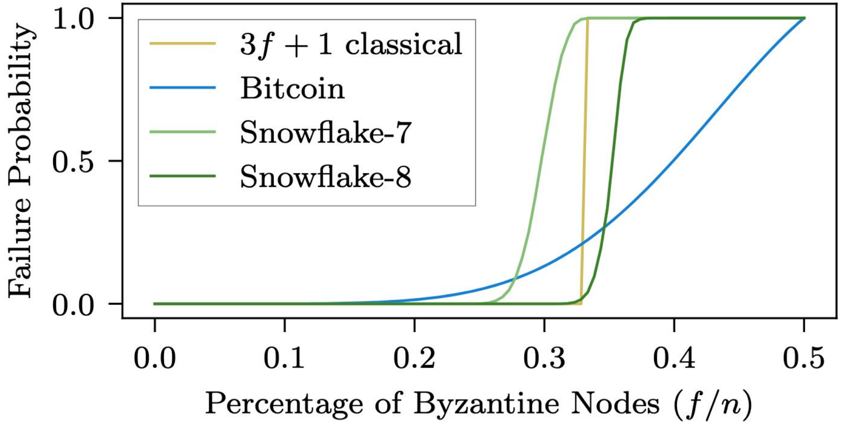
Snowflake: safety argument

```
def snowflake(me, col0):
  color = col0
  count = 0

  while True:
    if color == None: return None
    K = sample(N \ {me}, k)
    P = map(lambda v: query(v, color), K)

    for c in {RED, BLUE}:
      if P.count(c) >= alpha:
        count = 0 if color != c else count+1
        color = c

    if count > beta: return color
```

A: node a decides on **BLUE** at t_1

B: node b decides on **RED** at t_2

failure probability: $\mathbf{p(B \mid A) \leq \varepsilon}$



outline

- what is metastability?
- the Snow consensus family
  - Slush
  - Snowflake
  - **Snowball**
- Avalanche

Snowball: reduce random perturbations

```
def snowball(me, col0):
  color = lastc = col0
  conf[RED] = conf[BLUE] = 0
  count = 0

  while True:
    if color == None: return None
    K = sample(N \ {u}, k)
    P = map(lambda v: query(v, color), K)

    for c in {RED, BLUE}:
      if P.count(c) >= alpha:
        count = 0 if c != lastc else count+1
        lastc = c

        conf[c]++
        if conf[c] > conf[color]: color = c

    if count > beta: return color
```

new idea:

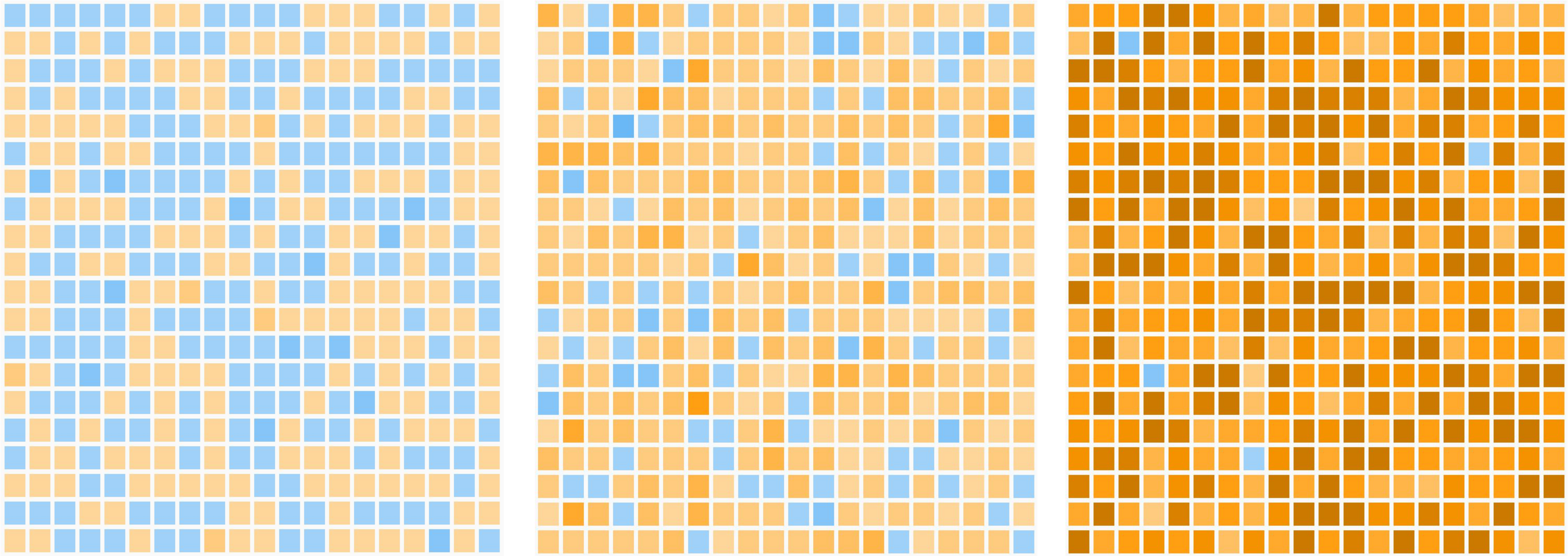
introduce history (confidence)

only change preference based on total confidence

new parameters:

beta: decision threshold

Snowball



https://avalabs.org/snow-bft-demo

outline

- what is metastability?
- the Snow consensus family
  - Slush
  - Snowflake
  - Snowball
- **Avalanche**

Avalanche: DAG-based digital payments

- observation: correct clients never double spend (attackers cannot forge signatures)
  - ⇒ safety and liveness guaranteed for virtuous transactions
  - ⇒ no liveness guarantee for rogue transactions

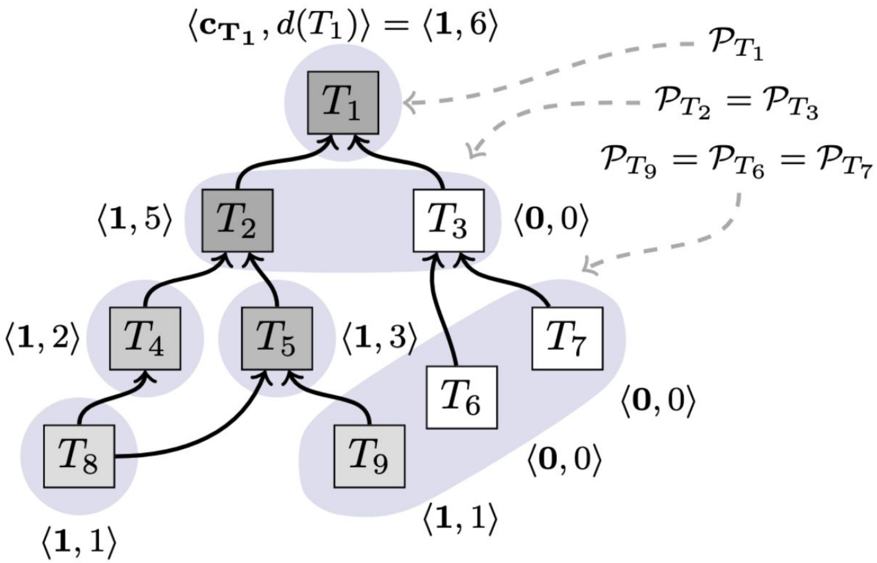
Scalable and Probabilistic Leaderless BFT Consensus through Metastability. (2019).

Avalanche: DAG-based digital payments

- maintain append-only DAG of transactions
  - ➜ application-defined parent-child relationship (not the same as UTXO!)
  - ➜ vertices belong to conflict sets
  - ➜ one Snowball instance for each conflict set ➜ choose one tx
- why DAG?
  - ➜ a vote on a vertex implicitly votes for all ancestors
  - ➜ confidence is derived from votes on predecessors
  - ➜ past decisions are harder to undo

Scalable and Probabilistic Leaderless BFT Consensus through Metastability. (2019).

Avalanche: DAG-based digital payments



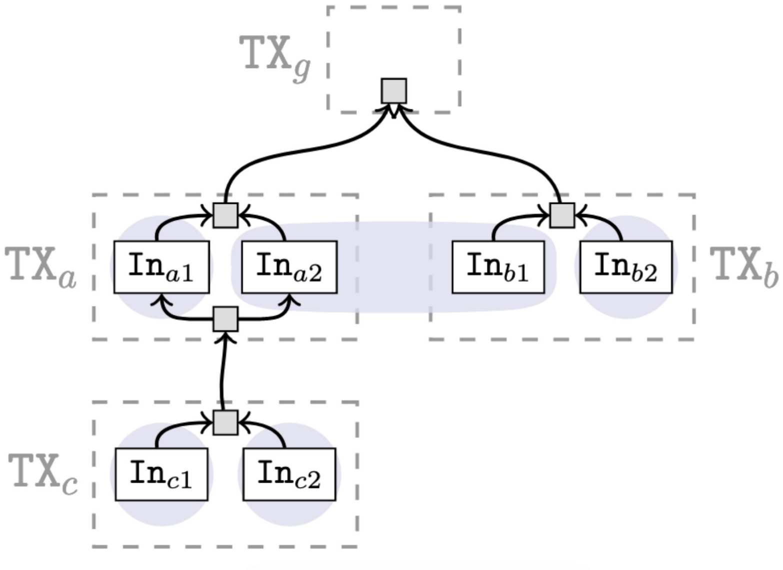
– vertices collect (immutable) 0/1 **chit** values using a one-time query

– **confidence** is derived from sub-DAG and grows as the DAG grows

– a vertex is **strongly preferred** if all its ancestors are the preferred one in their respective conflict sets

Scalable and Probabilistic Leaderless BFT Consensus through Metastability. (2019).

Avalanche: multi-input UTXOs



- financial transactions are embedded into Avalanche vertices, i.e. we have 2 DAGs
- each input corresponds to a single vertex
- tx accepted if all inputs are accepted

Scalable and Probabilistic Leaderless BFT Consensus through Metastability. (2019).

references

Scalable and Probabilistic Leaderless BFT Consensus through Metastability. (2019).
https://avalabs.org/snow-avalanche.pdf