UNIVERSITÄT BERN

Consensus: Theory and Practice

Christian Cachin
University of Bern

ICBC, June 2025

Some history

History – Consensus protocols

UNIVERSITÄT BERN

- **1980 until 2000:** Theory research – many theorems, no systems, no prototypes

The Byzantine Generals Problem · 385

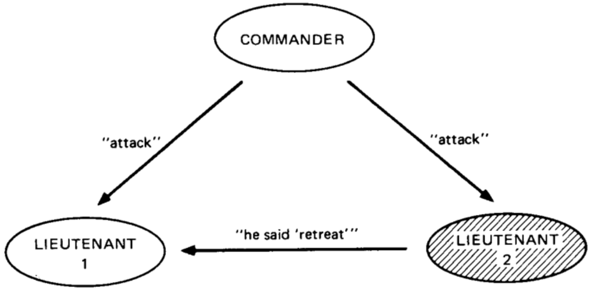


Fig. 1. Lieutenant 2 a traitor.

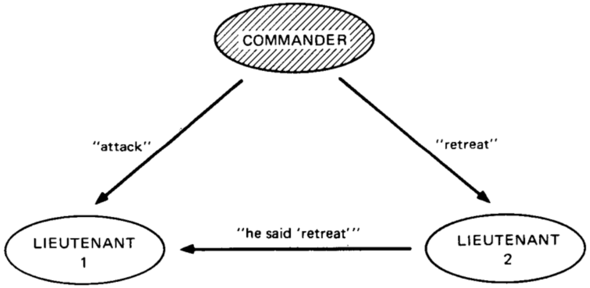


Fig. 2. The commander a traitor.

Unreliable Failure Detectors for Reliable Distributed Systems 255

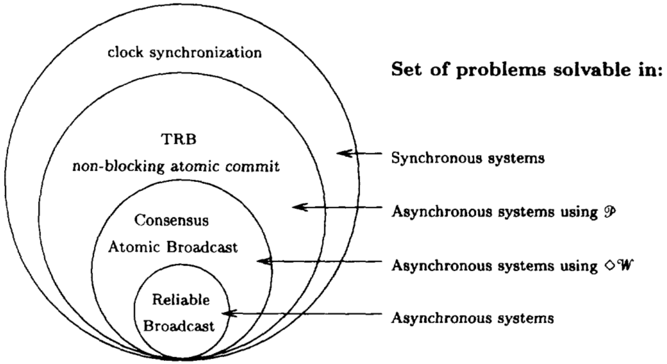


FIG. 9. Problem solvability in different distributed computing models.

234 T. D. CHANDRA AND S. TOUEG

Completeness	Accuracy: Strong	Accuracy: Weak	Accuracy: Eventual Strong	Accuracy: Eventual Weak
Strong	*Perfect* $\mathcal{P}$	*Strong* $\mathcal{S}$	*Eventually Perfect* $\diamond\mathcal{P}$	*Eventually Strong* $\diamond\mathcal{S}$
Weak	$\mathcal{Q}$	*Weak* $\mathcal{W}$	$\diamond\mathcal{Q}$	*Eventually Weak* $\diamond\mathcal{W}$

FIG. 1. Eight classes of failure detectors defined in terms of accuracy and completeness.

Impossibility of Distributed Consensus with One Faulty Process

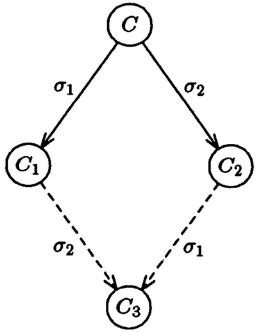
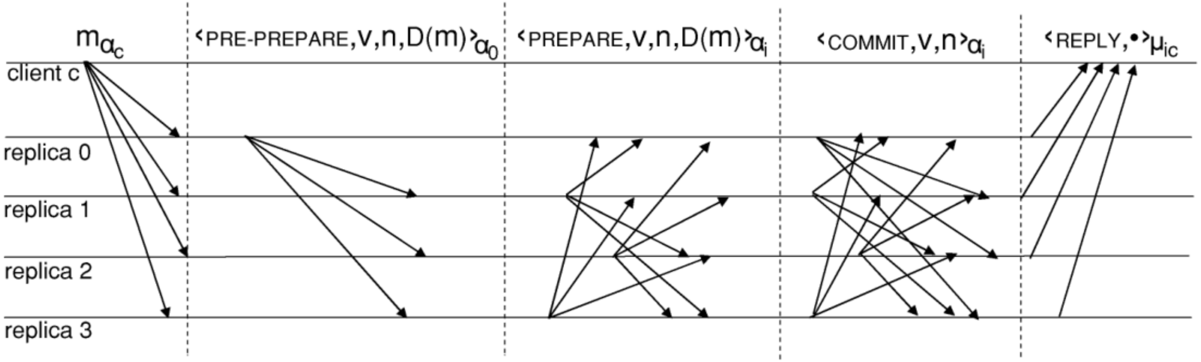


FIGURE 1

History – Consensus protocols

- **1980 until 2000:** Theory research – many theorems, no systems, no prototypes
- **2000 until 2010:** Systems research – many prototypes, no products

408 • M. Castro and B. Liskov



The Next 700 BFT Protocols

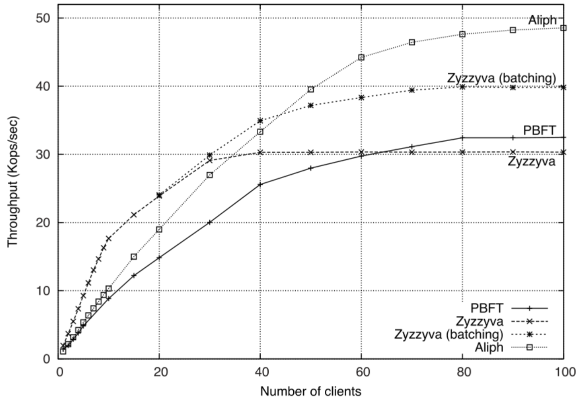


Fig. 8. Throughput for the 0/0 benchmark (f=1).

UNIVERSITÄT BERN

BFTW^3: Why? When? Where?
Workshop on Theory and Practice of Byzantine Fault Tolerance

Affiliated with DISC 2009

September 22, 2009
Elche, Spain

The workshop gathers researchers from both theory and systems communities and aims at understanding why the impressive research activity in the area of Byzantine fault-tolerance is not yet instantiated in practice. Has the moment for a wide deployment of BFT systems arrived, and if so, where BFT systems should be deployed in the first place?

Format

The workshop will consist of invited contributions. No published proceedings, the presentations may contain results that appeared or are going to appear elsewhere, work-in-progress reports, surveys and tutorials. A submission is expected to be a short (around two pages) abstract of the presentation.

UNIVERSITÄT BERN

BFTW^3: Why? When? Where?
Workshop on Theory and Practice of Byzantine Fault Tolerance

Affiliated with DISC 2009

September 22, 2009
Elche, Spain

However, there are few visible instantiations of these results in practical systems. Industrial software tends to ignore the BFT-related research and heads for less consistent but (apparently) simpler and more efficient solutions (e.g., [5, 16, 11]).

In this workshop, we discussed the state of the art in BFT systems, and tried to understand why BFT systems have not seen a widespread adoption, and what we could do to increase the chances of deploying BFT systems.

History – Consensus protocols

UNIVERSITÄT BERN

- **1985 until 2000:** Theory research – many theorems, no systems, no prototypes
- **2000 until 2010:** Systems research – many prototypes, no products
- **2010 onward:** Practice – deployment with cryptocurrencies

- **Today:** More theory research, more systems research, actual products, and practical deployment

Consensus protocols today

UNIVERSITÄT BERN

Overview

- Nine *models for consensus*
- Explore *the Snow protocol family of Avalanche*
- Concurrently, answer your questions

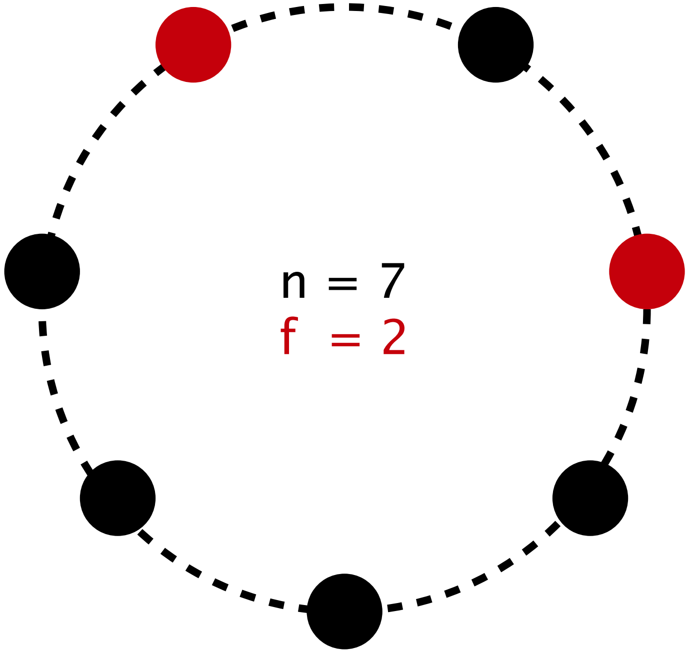
Nine models for consensus

UNIVERSITÄT
BERN

1 – Threshold trust (BFT)

UNIVERSITÄT BERN

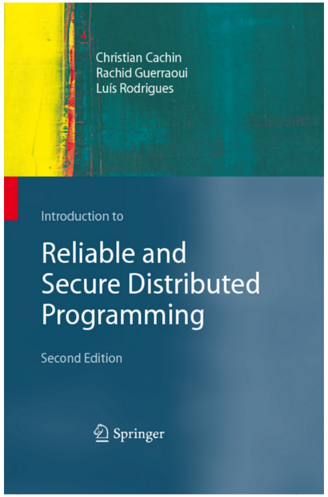
- Trust by numbers
  - n nodes total
  - f faulty (Byzantine) nodes



- Nodes are identified
  - Proof-of-Authority (PoA)

- Homogeneous and symmetric

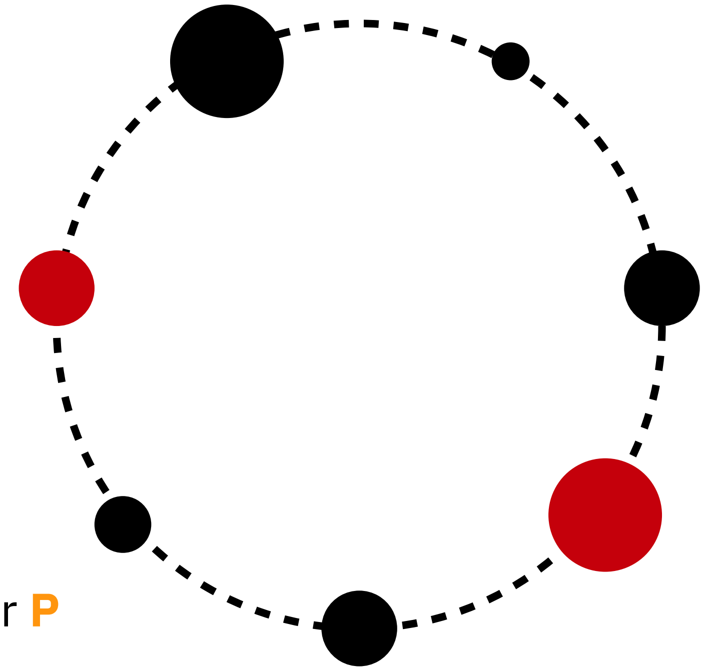
- Requires $n > 3f$



- Tendermint/Cosmos, Internet Computer (DFINITY), Hyperledger Fabric, VeChain, BNB SC, Hashgraph, TRON ...

2 – Generalized trust

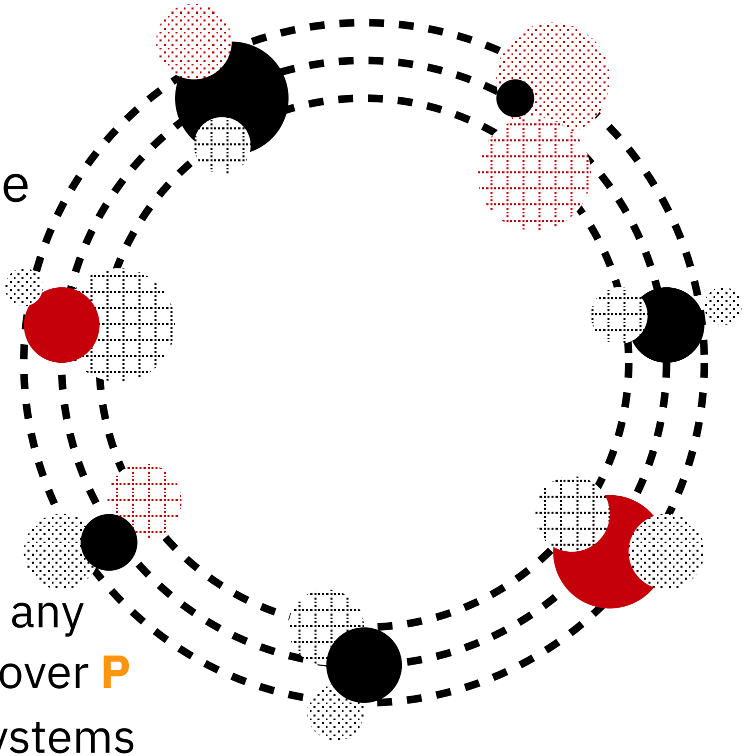
- Trust by generalized quorums
  - Set of nodes **P**
  - Fail-prone sets consisting of possibly Byzantine nodes
  - Byzantine quorum system



- Heterogeneous and symmetric
- Requires Q3-property
  - Any 3 fail-prone sets must not cover **P**
- Not used for consensus in any cryptocurrency (!)
  - But in distributed cryptography (Coinbase Vault)

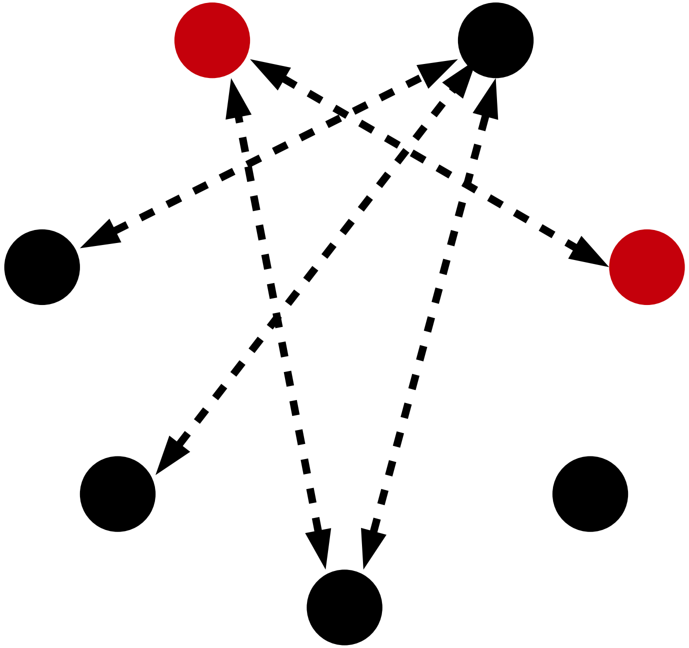
3 – Asymmetric trust

- Subjective generalized quorums
- Every node has its own fail-prone sets and quorum system on **P**
- Heterogeneous and asymmetric
- Requires B3-property
  - ∀ p, p' : any fail-prone set of p with any set of p' and any of both must not cover **P**
  - Consistent across nodes quorum systems
- Ripple, Stellar, [ACTZ24]



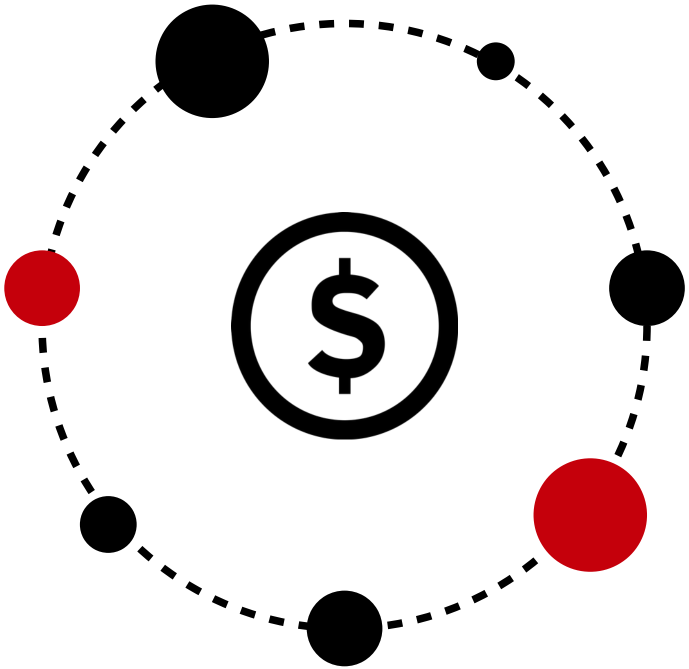
4 – Unstructured, probabilistic voting

- Random sampling of peers
- Exchange information and votes
- Often coupled with a DAG (directed acyclic graph) on transactions
- Avalanche, Conflux, IOTA-Tangle



5 – Stake-based voting

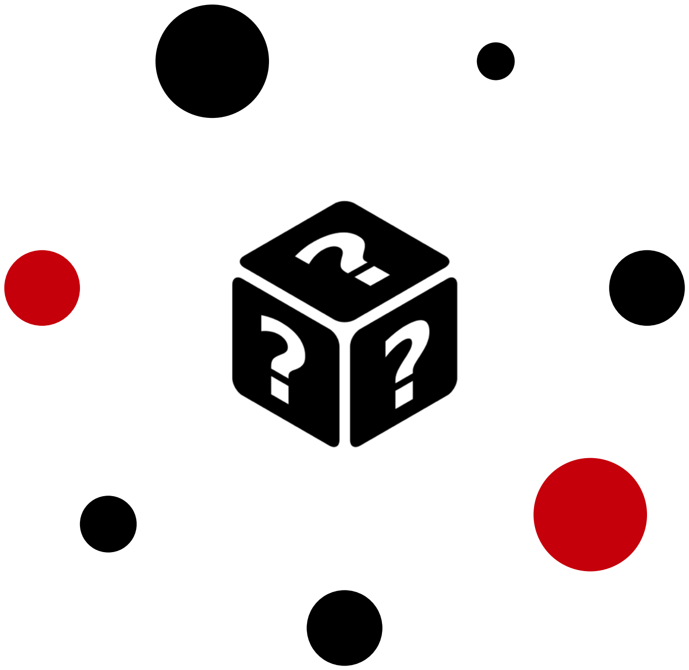
- Stake determines voting power
  - Including delegated stake (DPoS)
- Protocols generalized from symmetric voting (BFT)
- Slashing of invested stake upon detection of misbehavior
- Tendermint/Cosmos, EOS, NEO, Aptos, SUI, BNB SC ...



UNIVERSITÄT BERN

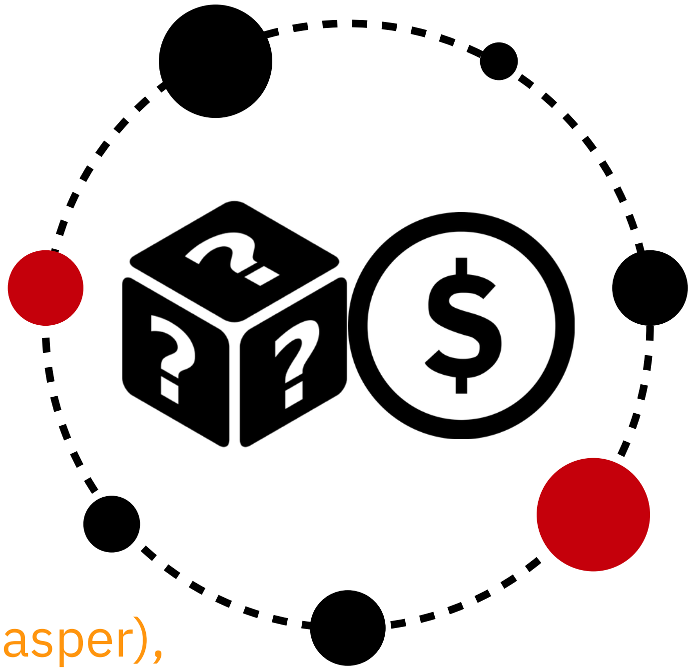
6 – Stake-based probabilistic choice

- Lottery according to stake
- Probabilistic leader election
- Cryptographic sortition using a verifiable random function (VRF)
- Cardano/Ouroboros ...



7 – Hybrid prob. choice and stake voting

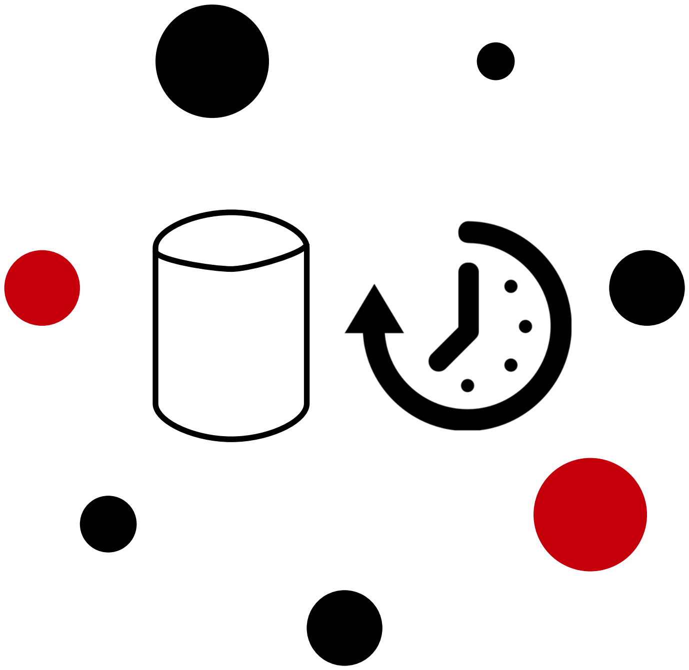
- Stake determines probability or voting power
- Mix of random choice with voting
- Slashing of invested stake upon detection of misbehavior
- Ethereum (LMD-GHOST & FFG-Casper), Polkadot (BABE & GRANDPA), Algorand ...



8 – Proof-of-space and proof-of-delay

UNIVERSITÄT BERN

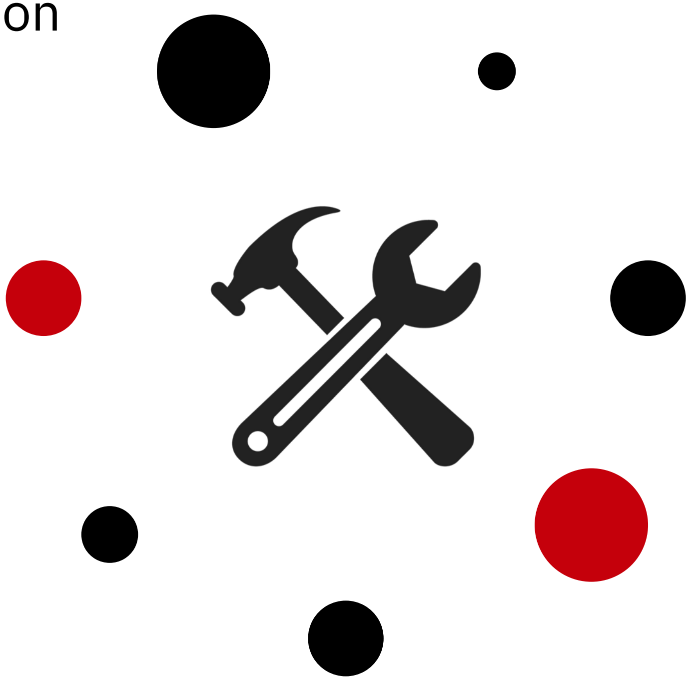
- Storage space as resource
- Cryptographic ZK proofs for storage at particular time
- Time delay to prove storage investment over time
- Filecoin, Chia, Storj ...



9 – Proof-of-work

UNIVERSITÄT
BERN

- Demonstrate invested computation
- Nakamoto consensus
- Bitcoin and variations,
  Litecoin, Dogecoin,
  Ethereum (1.0) and variations,
  Ethereum Classic,
  Monero, ZCash ...



Communication complexity

b
UNIVERSITÄT
BERN

Communication in consensus protocols

- Network of n nodes
- Tolerate f faulty nodes

- BFT protocols use Byzantine quorums: any set of $b > (n+f)/2$ nodes
  → $\Omega(n)$ messages per node
  → $\Omega(n^2)$ messages in the network

- With 1000s of nodes, this becomes infeasible!

How to reduce communication (1)

- Gossip messages logically
  - Communicate with $O(\log n)$ nodes, but send $\Omega(n)$ logical messages
  - Tendermint/CometBFT ...
    → Still $O(n^2)$ logical steps per node

- Actually gossip messages (probabilistic broadcast)
  - Bitcoin, Ethereum ...
    → $O(\log n)$ or $O(1)$ messages per node
  - Only probabilistic delivery guarantees
  - Cost of higher latency

How to reduce communication (2)

- Randomly chose a committee
  - Select committee of k among n nodes with unbiasable randomness (→ how?)
    → w.h.p. about k/3 nodes in committee are faulty
  - Let committee run consensus on block/transaction
  - Disseminate the result (→ how?)
  - Repeat! (Pick a fresh committee for next block ...)
  - Set $k = \log n$, then $O(\text{polylog } n)$ communication per node
  - Unbiasable randomness? E.g., cryptographic Verifiable Random Functions (VRFs)
  - Disseminate decisions? → Gossip

How to reduce communication (3)

UNIVERSITÄT BERN

- Random polls others and interactive updates
  - Ask a few others for their opinion (cf. committee)
  - Update opinion based on answers from others
  - Repeat this until "most" nodes have stabilized and converged on same opinion

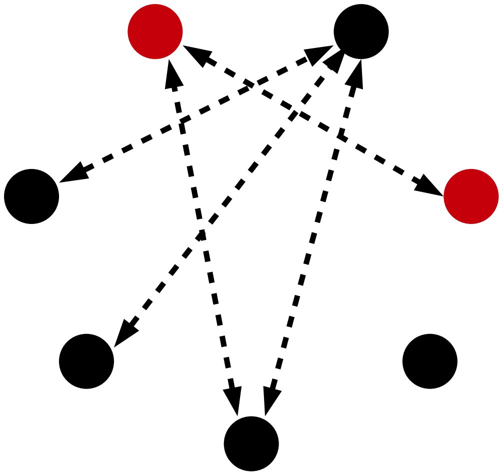
→ Consensus Dynamics: How a group of agents – such as robots, sensors, or decision-makers – interacting over a network can reach a common decision. [Wikipedia]

→ Avalanche Consensus

Snow and Avalanche consensus

UNIVERSITÄT BERN

Recent results with Ignacio Amores-Sesar & Philipp Schneider & Enrico Tedeschi



Avalanche

Avalanche

- Avalanche is a prominent layer-1 blockchain
  - AVAX cryptocurrency
  - Smart-contract platform
  - AVAX is in the top 15 by market cap

- Novel approach to consensus
  - "Snow family" of protocols:
    Slush → Snowflake → Snowball → [Snowman →] Avalanche
  - Introduced in a white paper 2019:
    Scalable and Probabilistic Leaderless BFT Consensus through Metastability (Yin, Sekniqi, van Renesse, Sirer)
  - Based on random sampling of peer nodes

Recent results [ACT22, ACT24]

UNIVERSITÄT BERN

- [ACS24]
  - Analysis of the consensus dynamics
  - Proofs for safety and liveness of (idealized) Snow protocols
  - Binary consensus

- [ACT22]
  - Detailed pseudocode of DAG-structured ledger consensus protocol
  - First independent analysis
  - Illustrated some problems and provided a solution
  - Generic broadcast (not quite atomic broadcast)

Avalanche network model

- Cryptocurrency and smart-contract platform
  - X-chain: eXchange (AVAX currency, other tokens)
  - P-chain: Platform (validator node management, staking)
  - C-chain: smart Contracts (EVM-compatible), with application-specific subnets

- n validator nodes
  - Each validator stakes 2000 AVAX (≈ 50'000 USD, Feb. 2025)
  - $n \approx 1400$ (Feb. 2025)
  - Throughput: ≈ 10 tps (on average); 50-100 tps (max. recorded); 4500 tps (max. claimed)

- Security
  - Tolerates faulty (Byzantine) nodes
  - Secure "only" against corruption of up to $\sqrt{n}$ nodes

Problem statement

- Consensus is binary
  - All nodes *propose* 0 or 1
  - All correct nodes have *stabilized* on the same value – or – they *decide* the same value

- Protocol operates in synchronous rounds
  - Number of rounds T
  - Security parameter β

- Randomized protocol
  - Every node sends and receives about O(k) messages per round, k small

- Termination
  - All correct nodes terminate after T rounds, except with probability negligible in β

Goals

- Fix k as small constant
  - $O(n)$ messages overall

- Number of rounds T
  - should be logarithmic in n
  - should be polynomial in β

- Related to the literature on dynamics of consensus
  - Overview by Becchetti, Clementi, Natale (SIGACT News, 2020)

Slush

Stabilization for consensus: Slush

- $b \in \{0,1\}$ // consensus on a bit

- **for** round = 1, ..., T **do**
  - pick k random parties, query them for their bit b
  - **if** at least α answers are b* **then** // $\alpha > k/2$

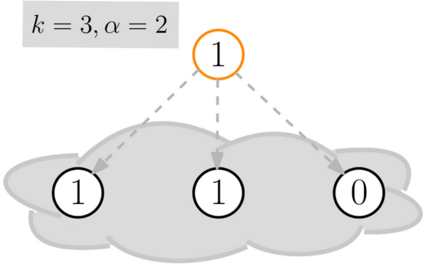
    $b \leftarrow b^*$
- decide(b) // after a fixed number of T rounds

Stabilization for consensus: Slush

- $b \in \{0,1\}$ // consensus on a bit

- **for** round = 1, ..., T **do**
  - pick k random parties, query them for their bit b
  - **if** at least α answers are b* **then** // $\alpha > k/2$

    b ← b*
- decide(b)

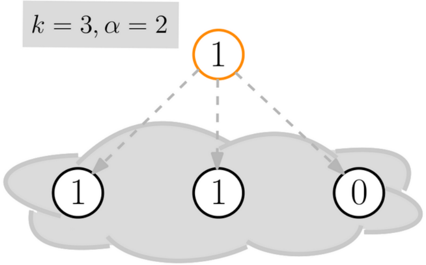


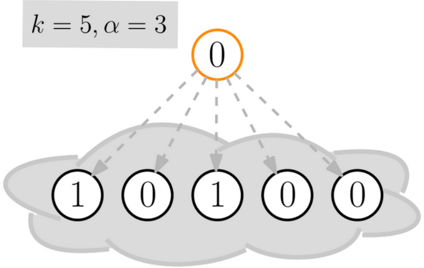
Stabilization for consensus: Slush

- $b \in \{0,1\}$ // consensus on a bit

- **for** round = 1, ..., T **do**
  - pick k random parties, query them for their bit b
  - **if** at least α answers are b* **then** // $\alpha > k/2$

    b ← b*
- decide(b)



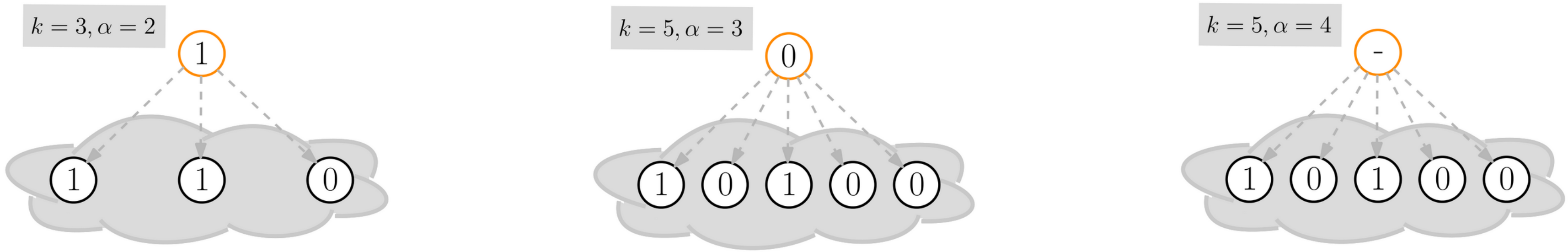


Stabilization for consensus: Slush

- b ∈ {0,1} // consensus on a bit

- **for** round = 1, ..., T **do**
  - pick k random parties, query them for their bit b
  - **if** at least α answers are b* **then** // α > k/2

    b ← b*
- decide(b)



How does Slush perform?

UNIVERSITÄT BERN

- Let p_i be fraction of nodes with opinion 1 in round i
- Let $\delta_i \in [-1, 1]$ be the expected "progress" towards consensus on 1
- For fixed k and α, progress δ_i is a function of p_i :

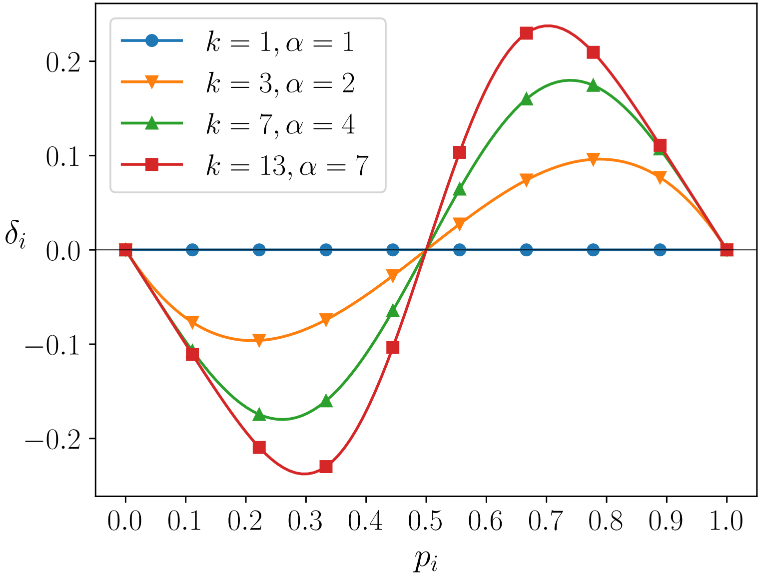
$$\delta_i = \delta(p_i) := \sum_{\ell=\alpha}^{k} \binom{k}{\ell} \left[p_i^\ell (1-p_i)^{k-\ell+1} - (1-p_i)^\ell p_i^{k-\ell+1} \right].$$

How does Slush perform?

UNIVERSITÄT BERN

- Let p_i be fraction of nodes with opinion 1 in round i
- Let $\delta_i \in [-1, 1]$ be the expected "progress" towards consensus on 1
- For fixed k and α, progress δ_i is a function of p_i :

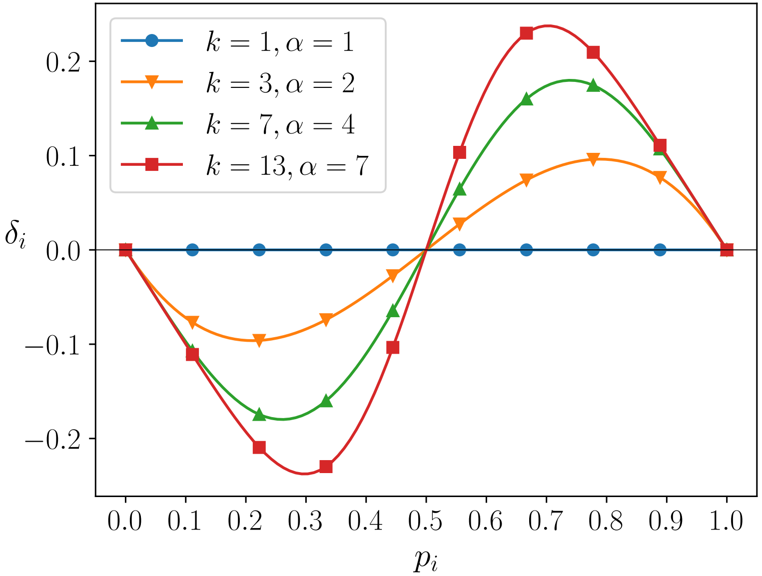
$$\delta_i = \delta(p_i) := \sum_{\ell=\alpha}^{k} \binom{k}{\ell} \left[p_i^\ell (1-p_i)^{k-\ell+1} - (1-p_i)^\ell p_i^{k-\ell+1} \right].$$

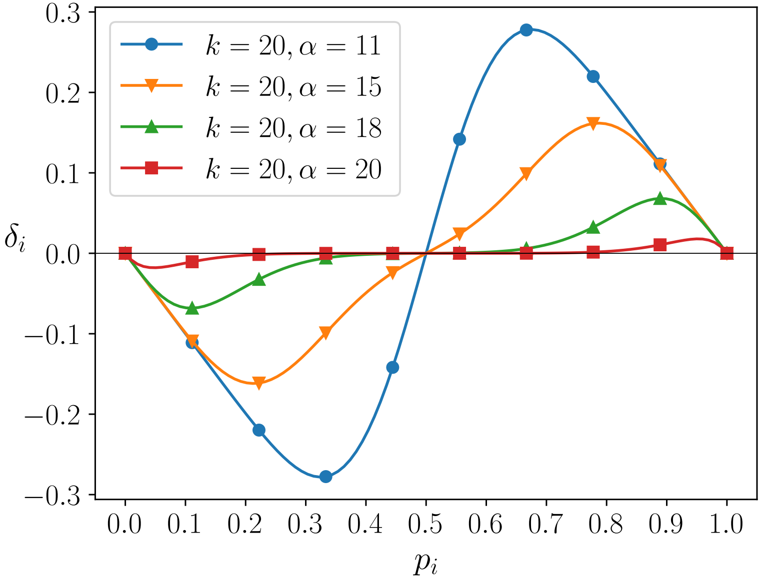


How does Slush perform?

- Let p_i be fraction of nodes with opinion 1 in round i
- Let $\delta_i \in [-1, 1]$ be the expected "progress" towards consensus on 1
- For fixed k and α, progress δ_i is a function of p_i :

$$\delta_i = \delta(p_i) := \sum_{\ell=\alpha}^{k} \binom{k}{\ell} \left[p_i^{\ell} (1-p_i)^{k-\ell+1} - (1-p_i)^{\ell} p_i^{k-\ell+1} \right].$$





Results on stabilization for Slush

- Theorem 1: For $k \geq 2$ and $\alpha = (k+1)/2$, Slush reaches stable consensus in

$$O(\log n + \lambda)$$

  rounds, with all but negligible probability in λ and up to $O(\sqrt{n})$ corrupted nodes.

- Theorem 2: For $k \geq 2$ and $k/2 < \alpha < k$, the expected number of rounds for Slush rounds to reach a stable consensus is

$$\Omega(\log n / \log k),$$

  with up to $O(\sqrt{n})$ corrupted nodes.

Snowflake

Decide after a stable period: Snowflake

- $b \in \{0,1\}$ // consensus on a bit
- counter ← 0
- **while** counter < β **do**
  - pick k random parties, query them for their bit b
  - **if** at least α answers are b* ≠ b **then** // α > k/2, Snowflake reset condition (+)
    - b ← b*
      counter ← 0
  - **else**
    counter ← counter + 1
- decide(b) // after β consecutive rounds with α-maj. for b

Decide more robustly: Snowflake+

UNIVERSITÄT BERN

- b ∈ {0,1} // consensus on a bit
- counter ← 0
- **while** counter < β **do**
  - pick k random parties, query them for their bit b
  - **if** at least α1 answers are b* ≠ b **then** // α1 > k/2, reset with simple majority
    b ← b*; counter ← 0
  - **if** at least α2 answers are b **then** // α2 > α1, count a strong majority
    counter ← counter + 1
  - **else**
    counter ← 0
- decide(b) // after β consecutive rounds with α2-maj. for b

Snowball is an earlier variant of Snowflake+

Some practical numbers

b UNIVERSITÄT BERN

- Number of nodes: $n \geq 500$
- Number of faulty nodes: $f \leq n/5 = 100$
- Number of sampled nodes: $k = 80$
- Simple majority: $\alpha 1 = 41$
- Strong majority: $\alpha 2 = 72$

- Practical insights [BBLOS24]:
  - Desired failure probability: $\varepsilon = 10^{-22}$ $\varepsilon = 10^{-14}$ $\varepsilon = 10^{-6}$
  - Required stabilization period: $\beta = 12$ $\beta = 8$ $\beta = 4$

Insight on Snowflake+

UNIVERSITÄT BERN

- Theorem 3: In Snowflake+ (and Snowball), even with a weak adversary, these two properties are mutually exclusive:

  1) Consensus holds with all but negligible probability (in β);

  2) Correct parties decide after polynomially many (in β) rounds.

Blizzard

A better tradeoff for consensus: Blizzard

UNIVERSITÄT BERN

- Blizzard changes the termination rule by considering more history

  c_0 counts number of rounds ever with an α-majority for 0

  c_1 counts number of rounds ever with an α-majority for 1

- Blizzard termination rule in Snowflake (+): stop when difference larger than t

  – **if** $| c_0 - c_1 | \geq t$ **then** ...

- Theorem 4: Blizzard reaches consensus with all but negligible probability (in β) and terminates in up to $O(\log n + \beta)$ rounds.

Snowman++

From consensus to (atomic) broadcast

UNIVERSITÄT BERN

- Consensus
  - Every node, once: propose(val v)
  - Every node, once: decide(val v)

- Reliable broadcasts
  - Every node, an arbitrary number of times: broadcast(msg m)
  - Every node, an arbitrary number of times: deliver(msg m)

- Atomic broadcast
  - All correct nodes deliver all messages in the same order (i.e., output the same sequence)

- Generic broadcast
  - All correct nodes deliver at least conflicted messages in the same order

Towards atomic broadcast

- Snowflake provides binary consensus

- Atomic broadcast on blocks of transactions

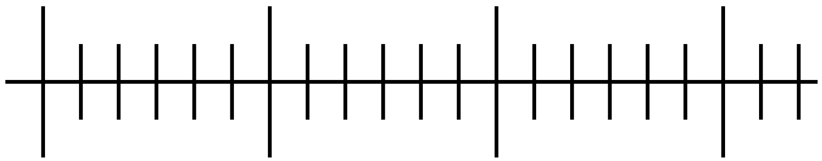
- Folklore idea (works in asynchronous model)
  - Proceed in (asynchronous) rounds
  - One consensus instance per round (all nodes propose, needs multivalued consensus)
  - Decides on a block for the round

- A simpler idea (requires synchronous model)
  - Operate in synchronized time slots
  - One node per slot proposes a block
  - One binary consensus decides on accepting slot of the proposer

Snowman++

- Live on Avalanche mainnet since ca. 2021

- Divide time into slots (30 seconds, initially), decide one block per slot
  - Divide slot into 6 windows



- Select one proposer node pseudo-randomly* for each window

- Run binary Snowflake+ consensus on block(s) from proposers
  - If all proposers stay silent, any node may propose a block

* Statically derived from seed in genesis block

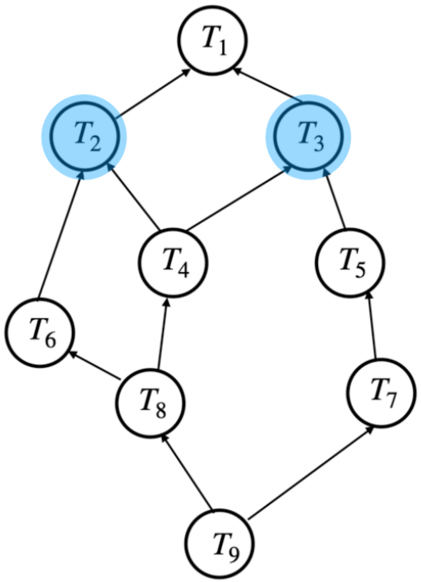
Back to another Avalanche

UNIVERSITÄT
BERN

DAG-ledger consensus in early Avalanche

UNIVERSITÄT BERN

- Was used in X-Chain of Avalanche until 2023
  - Extends consensus to a broadcast protocol
- Transactions form a DAG, a directed acyclic graph
- Transactions without dependencies (T2 and T3) may be delivered (accepted) in any order
  - Generic broadcast, parameterized by a conflict relation and weaker than atomic broadcast
- Transactions that conflict must be ordered



T2 and T2 independent

- Every transaction is decided with a Snowball-/Snowflake+ protocol

Avalanche DAG-ledger consensus

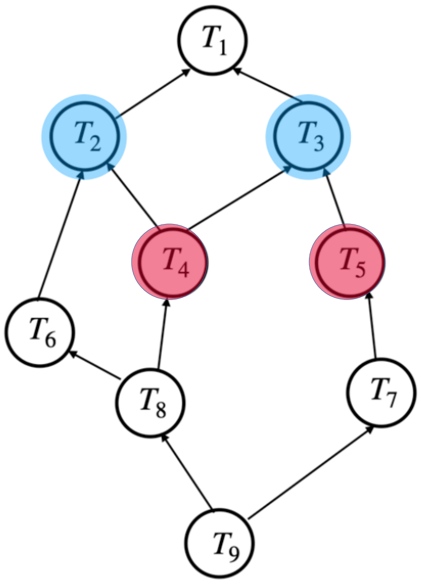
- **while** TRUE **do**
  - select some transaction T
  - pick k random parties and query them about T

  - **if** more than α positive results **then**

    update DAG: for every ancestor T' of T,
    increment counter(T') for acceptance
  - **else**

    update DAG: for every ancestor T' of T,
    reset (to 0) counter(T') for acceptance
  - **if** ($\exists$ T* that is not conflicting $\wedge$ counter(T*) $\geq$ β1) $\vee$
    ($\exists$ T* that is conflicting $\wedge$ counter(T*) $\geq$ β1) **then**

    deliver (output) T



Conflicting tx can come to exist in the DAG.

Referencing them cleverly can delay acceptance of innocent tx.

Analysis of DAG-ledger consensus [ACT22]

b UNIVERSITÄT BERN

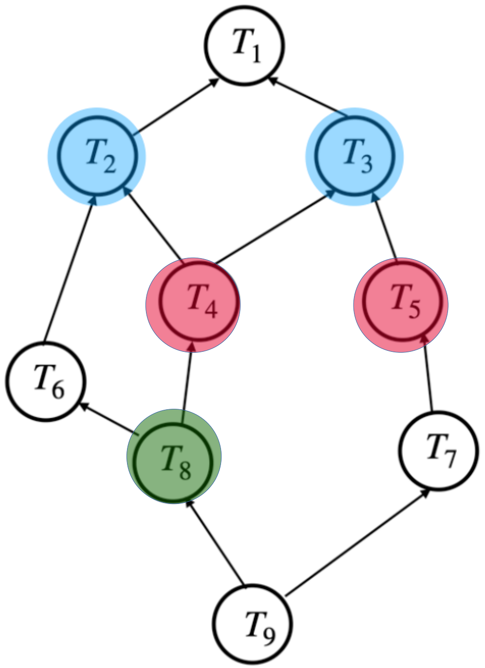
- Detailed pseudocode of Avalanche protocol

- Identified a liveness problem
  - Adversary A may delay acceptance of a victim transaction arbitrarily
  - DAG makes acceptance of transactions depend on each other
  - Subsequent transactions may delay acceptance of already existing (correct) transactions

- For other reasons, Avalanche abandoned the DAG protocol on the X-chain in March '23

A liveness issue in DAG-ledger consensus

UNIVERSITÄT BERN



- Adversary A may delay acceptance of a victim transaction arbitrarily
  - A submits conflicting transactions, e.g., T4 and T5
  - Waits until T4 is preferred and both referenced by other transactions
  - To attack victim transaction T8, A submits transactions that reference T8 and T5
  - T4 is preferred, but T8 and T5 conflict with T4
  - Causes counter of T8 to be reset to 0 arbitrarily often
  - Then T8 is delayed forever

Conclusion

- Byzantine-tolerant consensus protocols are here to stay
  – Models are more important than protocols

- Avalanche: Efficient probabilistic protocols with novel ideas
  – Interesting consensus dynamics

- Links
  – Web: https://crypto.unibe.ch/
  – Blog: https://cryptobern.github.io/
  – Bluesky: @cryptobern.bsky.social

Thanks

b UNIVERSITÄT BERN

- This work has been supported by
  - Swiss National Science Foundation (SNSF);
  - IC3 – The Initiative for CryptoCurrencies and Contracts;
  - Avalanche, Inc.;
  - Donations from Sui Foundation, Stellar Development Foundation, and more.

- Links
  - Web: https://crypto.unibe.ch/
  - Blog: https://cryptobern.github.io/
  - Bluesky: @cryptobern.bsky.social

References

References 1

UNIVERSITÄT BERN

- [ACT22] Amores-Sesar, I., Cachin, C., & Tedeschi, E. (2022). When is Spring coming? A Security Analysis of Avalanche Consensus. Proc. 26th International Conference on Principles of Distributed Systems (OPODIS), 10:1–10:22. https://doi.org/10.4230/LIPIcs.OPODIS.2022.10

- [ACS24] Amores-Sesar, I., Cachin, C., & Schneider, P. (2024). An Analysis of Avalanche Consensus. In Y. Emek (Ed.), Proc. Structural Information and Communication Complexity (SIROCCO) (Vol. 14662, pp. 27–44). Springer. https://doi.org/10.1007/978-3-031-60603-8_2

References 2

UNIVERSITÄT BERN

- [BBLOS24] Aaron Buchwald, Stephen Buttolph, Andrew Lewis-Pye, Patrick O'Grady, Kevin Sekniqi: Frosty: Bringing strong liveness guarantees to the Snow family of consensus protocols. CoRR abs/2404.14250 (2024)

- [BBLS25] Aaron Buchwald, Stephen Buttolph, Andrew Lewis-Pye, Kevin Sekniqi: Snowman for partial synchrony. CoRR abs/2501.15904 (2025)

- [G21] Patrick O'Grady: Snowman++: Congestion control for Snowman VMs https://github.com/ava-labs/avalanchego/blob/master/vms/proposervm/README.md