Asynchrony-Resilient Sleepy Total-Order Broadcast Protocols

Francesco D'Amato
Ethereum Foundation

Giuliano Losa
Stellar Development Foundation

Luca Zanolini
Ethereum Foundation

Dynamically-available protocols tolerate large-scale correlated (benign) failures in blockchains networks

This means protocols that:

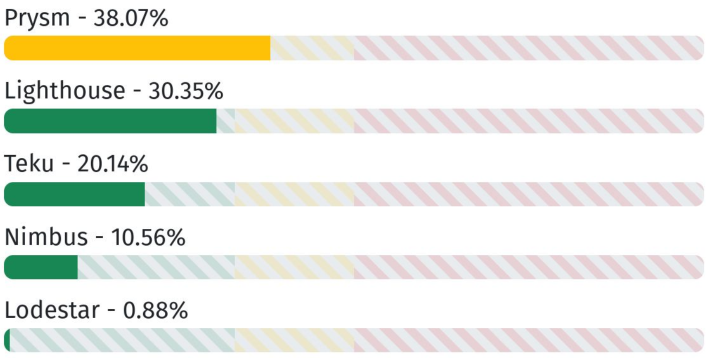
- Have a known list of participants
- But, can tolerate participants unpredictably going offline at any time (and even 99% of them)
- Also tolerate malicious (Byzantine) failures

Dynamically-available protocols are deployed in e.g. the Ethereum and Cardano blockchains

Example in the wild: software bug in Ethereum

- Ethereum promotes the use of diverse software implementations to avoid correlated failures
- But, in May 2023, a bug affected two implementations (Prysm+Teku) and roughly 60% of the participants went offline for 25 minutes
- The system kept working and applications were not affected
- Traditional BFT consensus uses fixed-sized quorums and would get stuck if > ⅓ crash

Consensus implementations on Ethereum

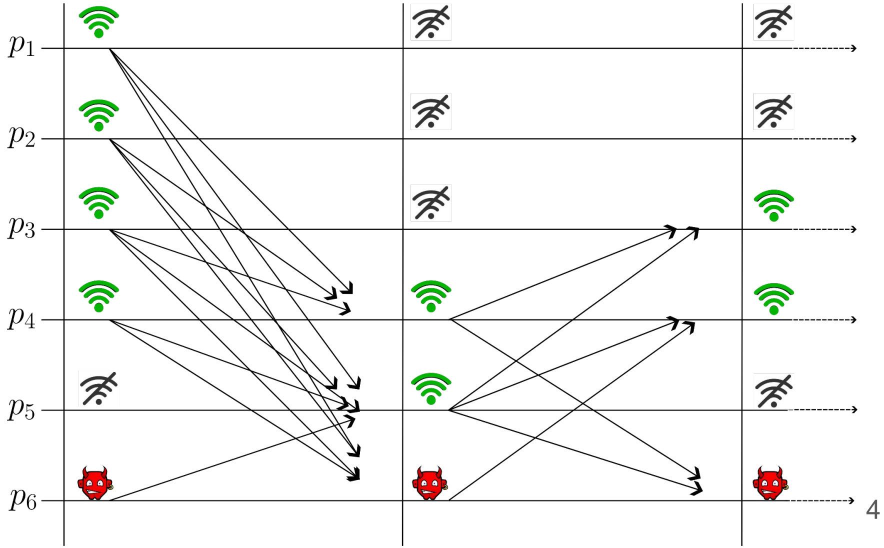


Source: https://clientdiversity.org/#distribution

The sleepy model* captures key aspects of dynamic availability

Example with β=1/2

- Participants are known but, each round, some may be offline
- Synchronous, reliable network
  - Message delay < 1 round
- Each round, less than a fraction β of the online participants are malicious
  - Adversary is constant or growing
- In practice, Ethereum uses real-time intervals of 12 seconds



*Rafael Pass and Elaine Shi. "The sleepy model of consensus." Advances in Cryptology–ASIACRYPT 2017

Drawback: the safety of dynamically-available protocols depends on synchrony

- All safety guarantees are lost if the network is not synchronous
  - Dynamically-available protocols use relative thresholds
  - Intersection arguments depend on messages from all well-behaved participants being reliably received by all
- In general, this is expected: eventually-synchronous, dynamically available consensus is impossible

  See Theorem 7.2 in: Lewis-Pye Roughgarden, Permissionless Consensus. arXiv preprint arXiv:2304.14701

Contribution: methodology to modify existing protocols to survive bounded periods of unreliable communication

Poor solution: slow down the protocol

- Use an extremely conservative round duration, e.g. not 12 seconds but 1 minute
- This slows down the protocol proportionally to the increase in round duration

E.g. 12 seconds to 1 minute: 5x slowdown

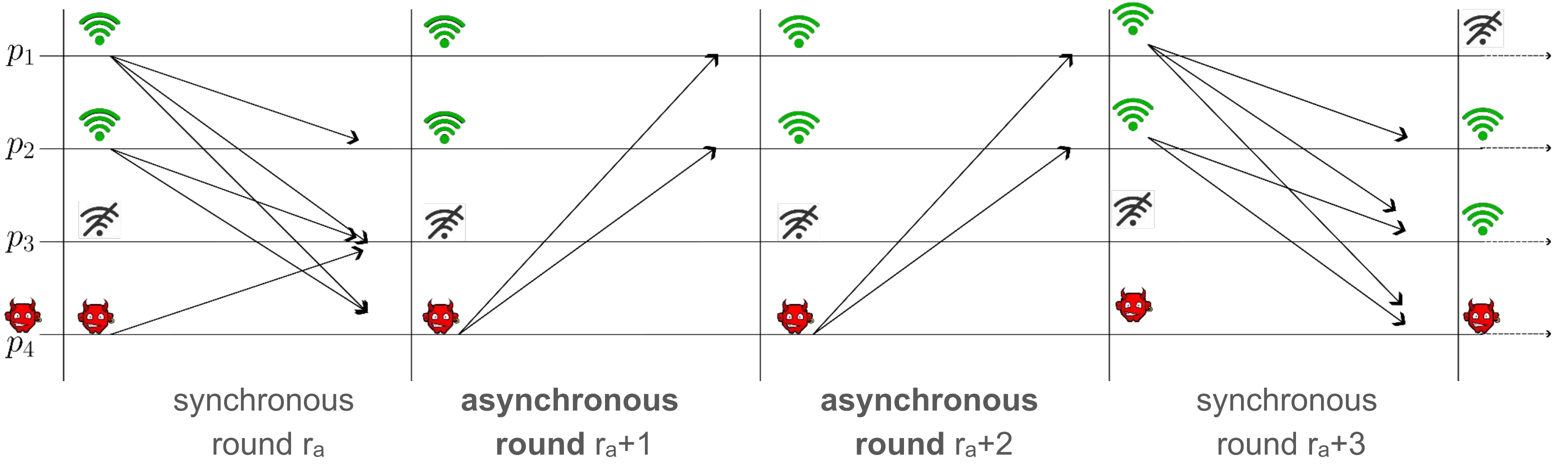
This paper

- Keep round duration the same to maintain performance
- Accept that, in rounds occurring during asynchronous periods, message delivery may be fully adversarial
- Modify existing protocols to keep them safe during asynchronous rounds

The sleepy model with an asynchronous period

- We assume a single asynchronous period spanning rounds $[r_a+1, r_a+\pi]$
- Message delivery in asynchronous rounds is fully under adversarial control

Examples with $\pi=2$:



Goal: Asynchrony-resilient Total-Order Broadcast

Total order broadcast

Processes add blocks to a growing sequence called a log. They *deliver* growing logs

Safety: for every two delivered logs, one is a prefix of the other

Liveness: if all processes get a block b as input, then (with non-zero probability) eventually a log containing b is delivered

Asynchrony-resilience conditions

During asynchrony ($[r_a+1, r_a+\pi]$)

- Delivered logs may conflict
- Processes that were online in r_a do not revert any log delivered before r_a
- No progress guarantees

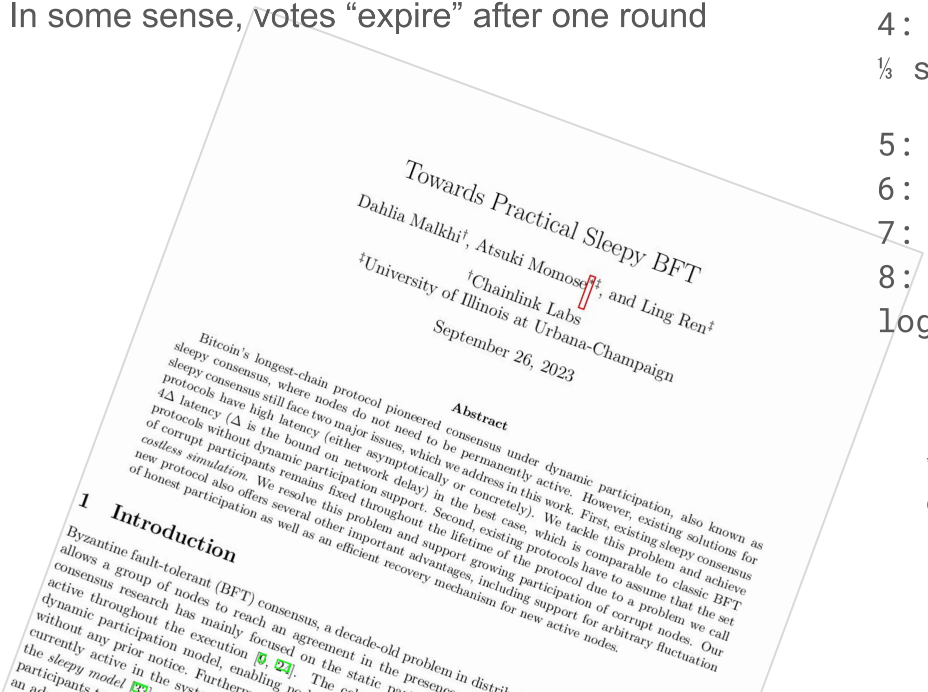
After asynchrony ($r_a+\pi+1$ and after)

- Newly delivered logs extend the logs delivered before r_a
- Newly delivered logs never conflict
- Progress guarantees resume

Example: ⅓-resilient total-order broadcast with the MMR protocol

Key observation: processes vote for logs and take action based only on votes cast in the previous round

In some sense, votes "expire" after one round


Towards Practical Sleepy BFT

Dahlia Malkhi†, Atsuki Momose‡, and Ling Ren‡

†Chainlink Labs

‡University of Illinois at Urbana-Champaign

September 26, 2023

Abstract

Bitcoin's longest-chain protocol pioneered consensus under dynamic participation, also known as sleepy consensus, where nodes do not need to be permanently active. However, existing solutions for sleepy consensus still face two major issues, which we address in this work. First, existing sleepy consensus protocols have high latency (either asymptotically or concretely). We tackle this problem and achieve 4Δ latency (Δ is the bound on network delay) in the best case, which is comparable to classic BFT protocols without dynamic participation support. Second, existing protocols have to assume that the set of corrupt participants remains fixed throughout the lifetime of the protocol due to a problem we call *costless simulation*. We resolve this problem and support growing participation of corrupt nodes. Our new protocol also offers several other important advantages, including support for arbitrary fluctuation of honest participation as well as an efficient recovery mechanism for new active nodes.

1 Introduction

Byzantine fault-tolerant (BFT) consensus, a decade-old problem in distri
allows a group of nodes to reach an agreement in the presen
consensus research has mainly focused on the static pa
active throughout the execution [9, 21]. The
dynamic participation model, enabling no
without any prior notice. Furthe
currently active in the sys
the *sleepy model* [3
participants
an ad

```
time (2k+1)Δ:
receive votes sent at time 2kΔ
vote for maximal log with > ⅔ support
propose extension of a maximal log with  > ⅓ support

time (2k+2)Δ:
receive votes sent at time (2k+1)Δ
deliver maximal log with > ⅔ support
vote for a proposal* extending a maximal log with > ⅓ support
```

*a probabilistic scheme ensures that all well-behaved processes extend the same "good" log with probability 1/3

We make MMR asynchrony-resilient using a vote-expiration period of η≥1 rounds

Protocol modifications

- We change how processes count votes
- For each process, we count the latest vote it cast no later than η rounds ago
  - i.e. votes expire only after η rounds
  - In vanilla MMR we have η=1
- The protocol otherwise remains unchanged

If η>π, we achieve asynchrony-resilience

Older votes prevent reverting logs delivered before asynchrony (assuming limited adversarial growth)

Possible disagreement on new blocks added during asynchrony, but

Normal protocol operation resumes after asynchrony (if enough processes stick around)

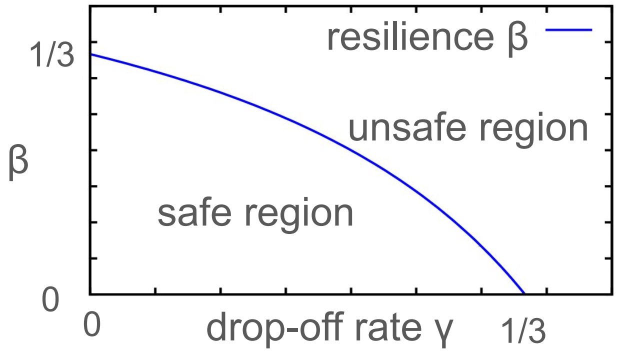
There’s a catch: the expiration period reduce resilience during synchrony

Drop-off rate γ: fraction of the well-behaved processes that were online during the expiration period and are no longer online

Resilience decreases with the drop-off rate: above the line, safety violations are possible

With a drop-off rate > ⅓, we lose adversarial resilience

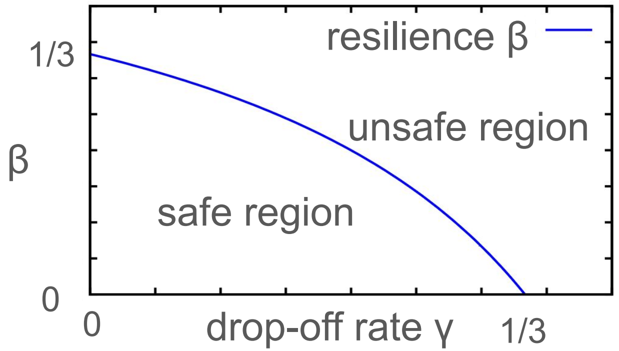
Intuitively: the adversary can use stale messages to its advantage, and so we must count stale messages as adversarial



If the drop-off rate cannot exceed ⅓ even during synchrony, have we not lost dynamic availability?

Not really! If the drop-off rate exceeds ⅓ then

- We are still safe if there is adversarial behavior
- We lose safety under adversarial behavior but:
  - Older votes prevent reverting logs delivered before the drop-off event
  - The protocol recovers after the expiration period
  - Safety-sensitive applications can choose to wait out the expiration period
- We temporarily lose progress guarantees



Asynchrony-resilient MMR achieves a new tradeoff

Dynamically-available consensus	**Asynchrony-resilient MMR**	Partially-synchronous consensus
Tolerates arbitrarily fluctuating participation (even 99%)	Live under arbitrarily fluctuating participation	A fixed number of processors must remain available (e.g. 2f+1)
Only safe under synchrony	Delivered prefixes are safe for η asynchronous rounds	Safe during asynchrony
	During synchrony, full safety only under bounded drop-offs or no adversarial behavior	
	The protocol recovers safety and liveness after η “good” rounds	

Conjecture: the methodology applies to most existing dynamically-available protocols

Including:

- Momose and Ren. **Constant Latency in Sleepy Consensus**. CCS 2022.
- Malkhi, Momose, and Ren. **Towards Practical Sleepy BFT**. CCS 2023.
- Losa and Gafni. **Brief Announcement: Byzantine Consensus Under Dynamic Participation with a Well-Behaved Majority**. DISC 2023
- D’Amato and Zanolini. **Streamlining Sleepy Consensus: Total-Order Broadcast with Single-Vote Decisions in the Sleepy Model**. Arxiv:2310.11331
- D’Amato and Zanolini. **Recent Latest Message Driven GHOST: Balancing Dynamic Availability With Asynchrony Resilience**. arXiv:2302.11326