Efficient algorithms for CBC Casper

IC3 2019

Vlad Zamfir, Aditya Asgaonkar, Nate Rush, Carl Beekhuizen
Hsiao-Wei Wang & Paul Hauner

LMD GHOST

Latest Message Driven, Greediest Heaviest Observed Sub-Tree

Chooses the head of a chain by:

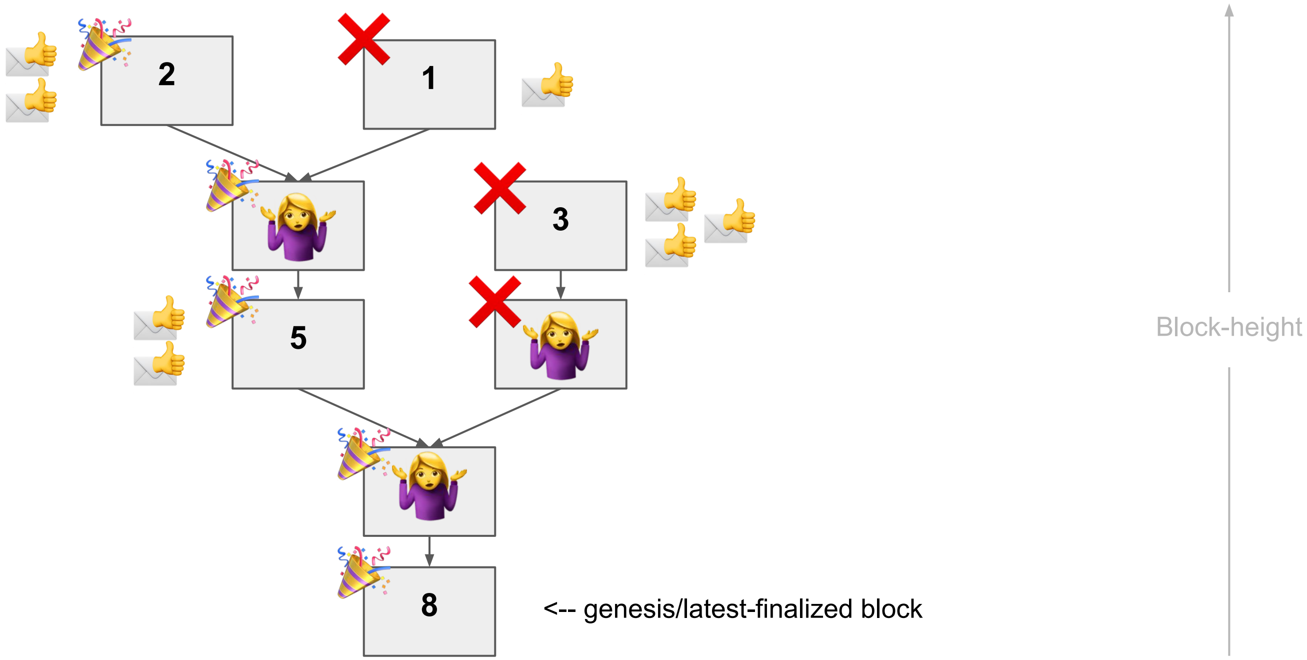
1. Only considering the “latest” (highest block) messages from validators.
2. Walking up the tree, greedily choosing the branch with the most votes (votes may be weighted).

Why optimize?

We want to run the algorithm frequently:

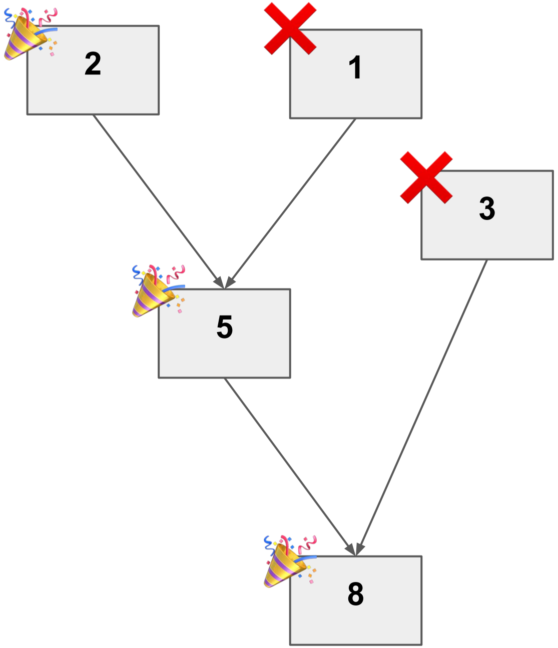
- CBC Casper protocols test the validity of previous block pointers
  - In this case, we use the LMD GHOST fork choice rule.
- Eth2 might be more secure if LMD GHOST is used to validate block pointers.

But, some of these blocks were unnecessary...



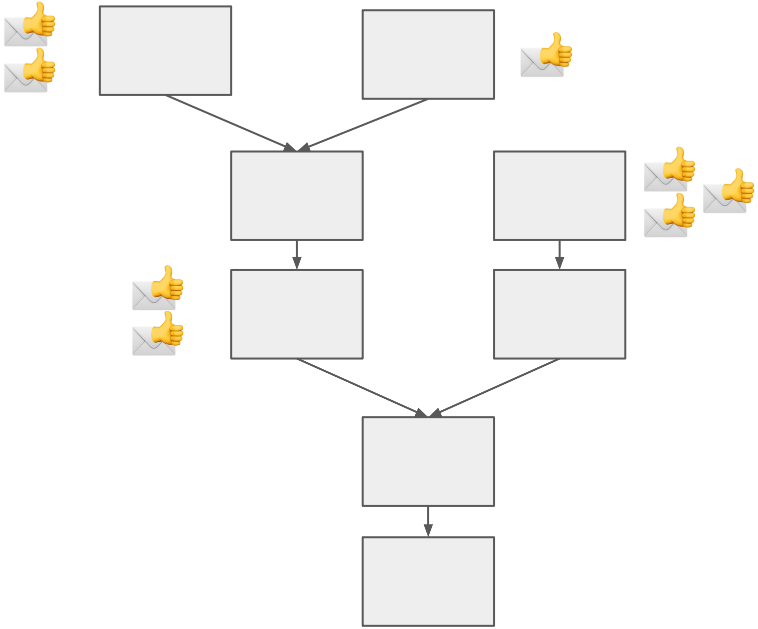
👍 = validators latest message attesting to block

Reduced-Tree LMD GHOST 🥳

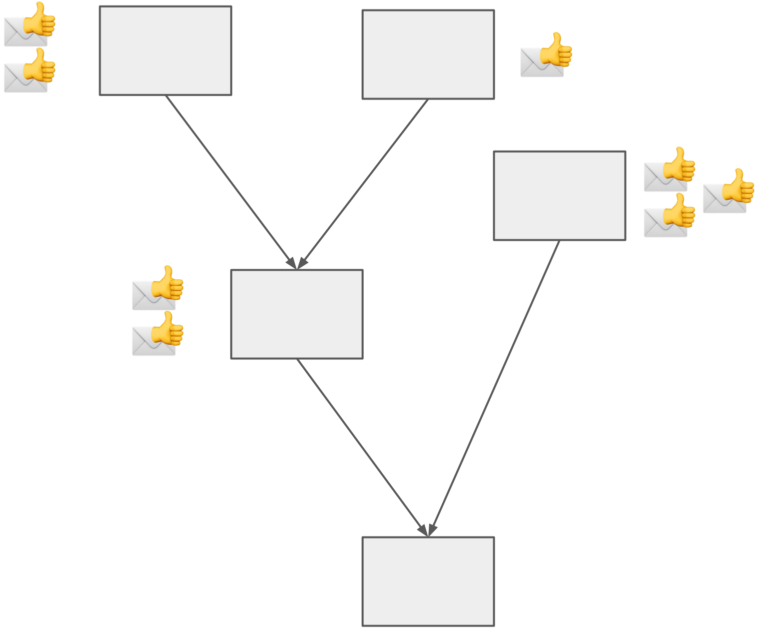


Credits to Nate Rush for original concept.

To summarize



-- becomes -->

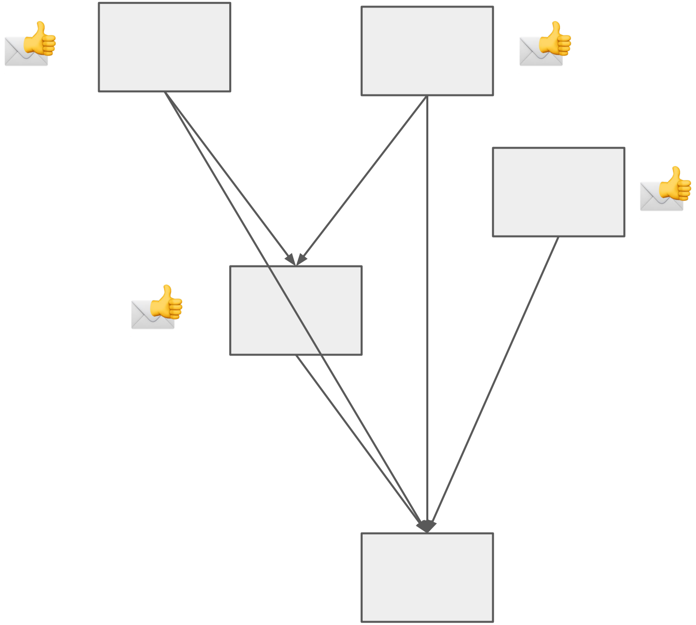


Maintaining the tree

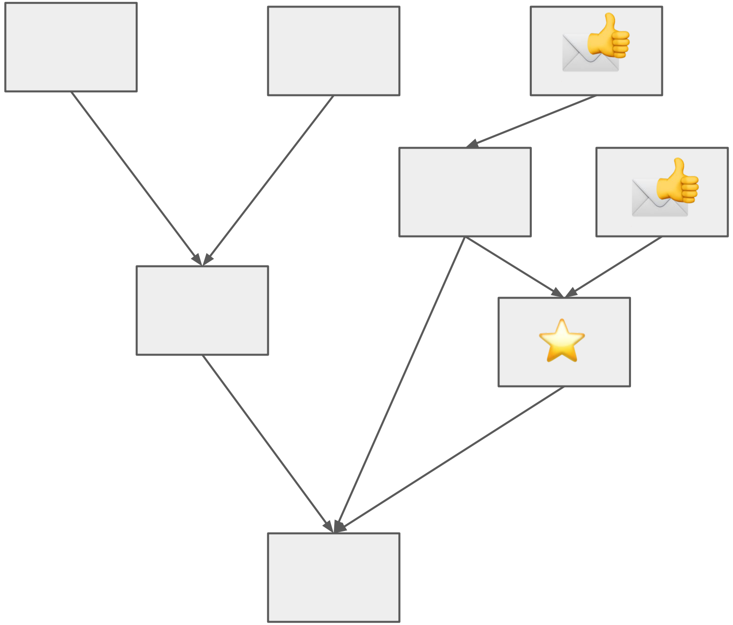
Required operations:

- Update latest message
    - Add latest message
    - Remove old message
- Prune for finalization
- Find head

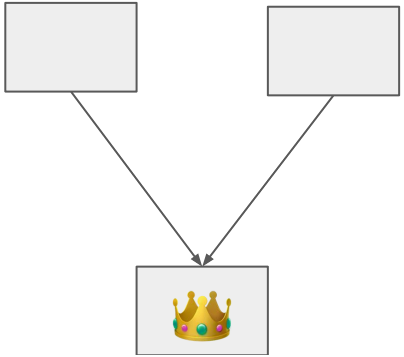
Operation: remove latest messages



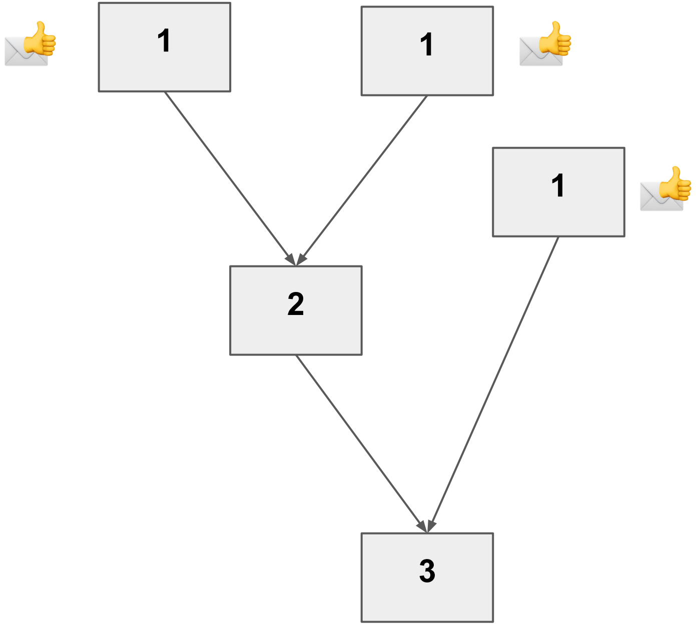
Operation: add latest message



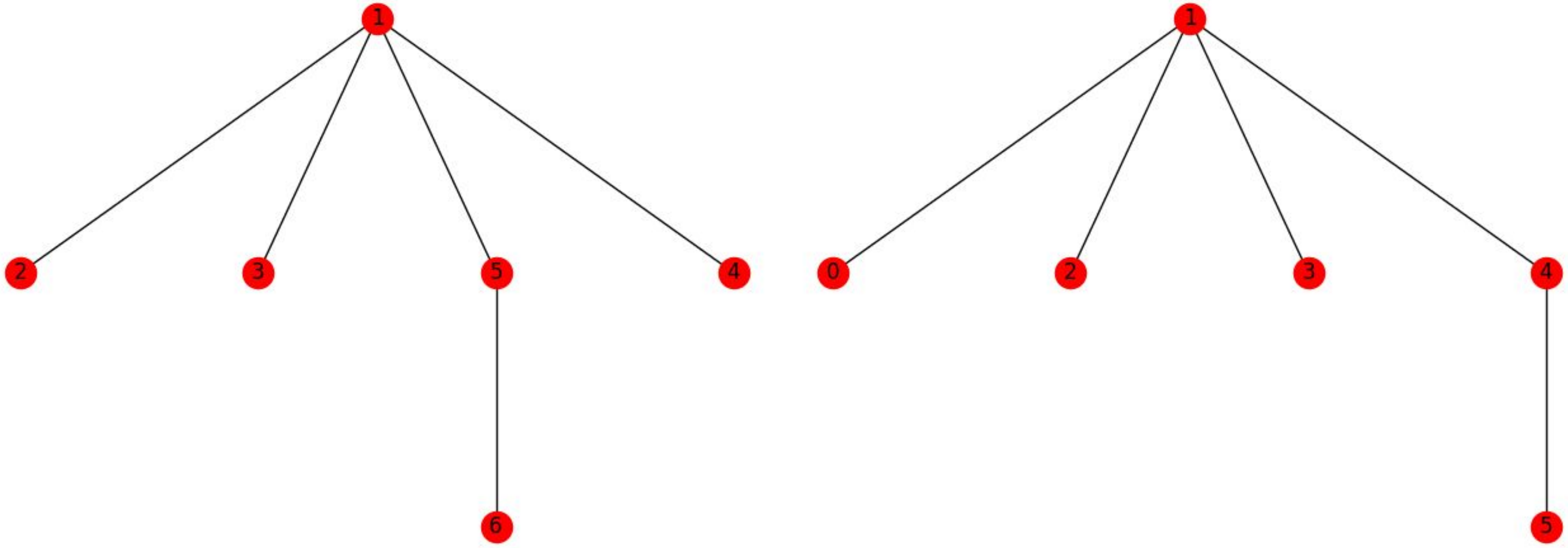
Operation: prune after finalization



Operation: find head



Normal vs Compressed Tree



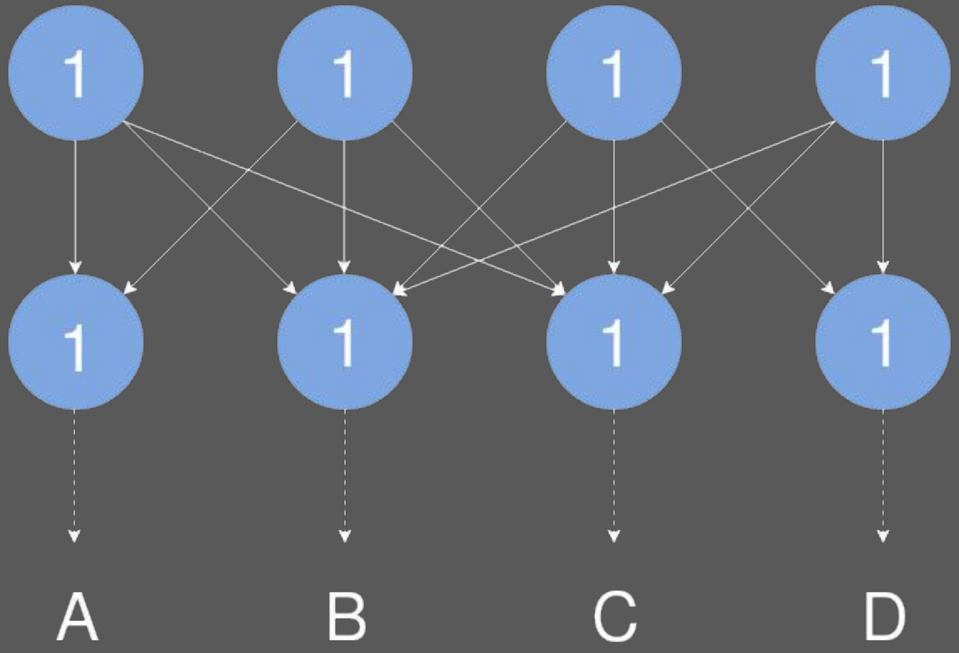
Reduced Tree Characteristics

	Naive LMD GHOST	Reduced Tree
Find Head	O(B)	**O(V)**
Add Message	O(1)	**O(V)**
Remove Message	O(1)	**O(log V)**

Reduced Tree vs Bitwise LMD GHOST

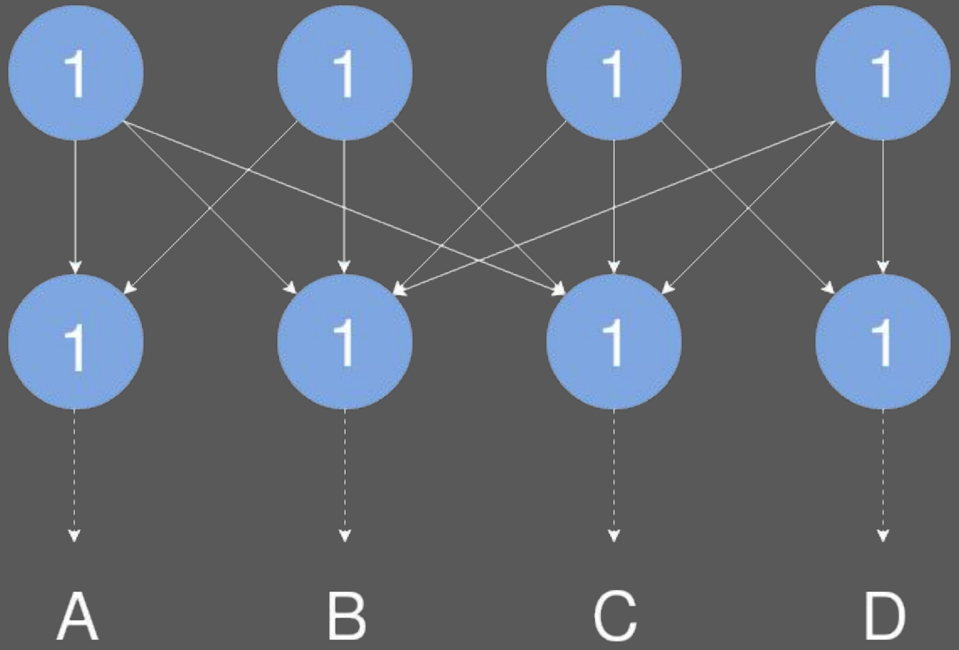
	Naive LMD GHOST	Previous State of the Art	Reduced Tree
Find Head	O(B)	O(V*log(h))	**O(V*log(V))**

Finality Inspector



Finality is achieved on a value when it is impossible for any node to legitimately produce a message that proposes a contradictory value

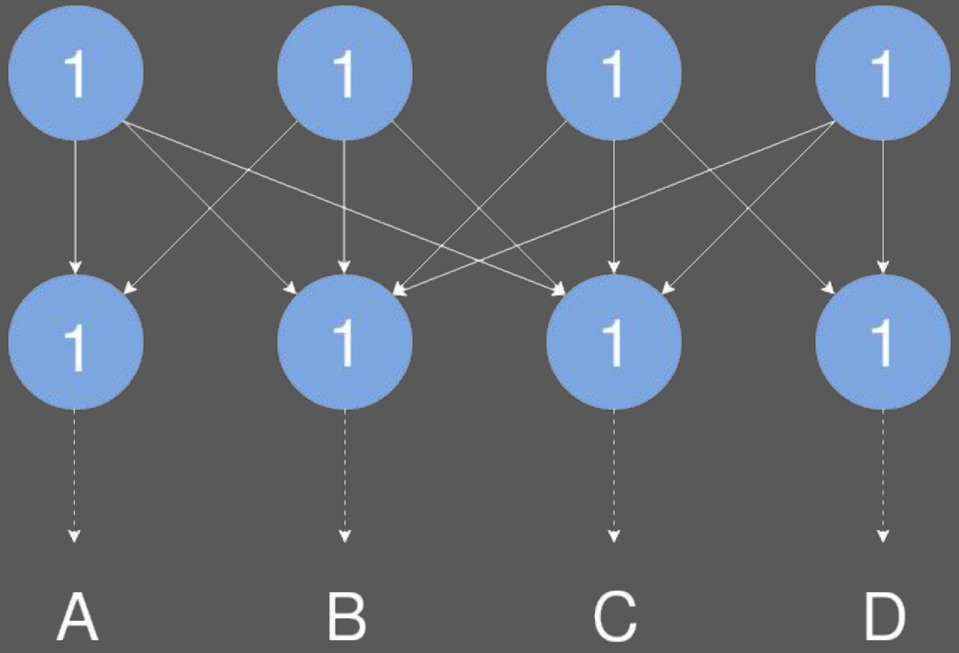
Finality Inspector



In fact, we can determine the “finality score” for a given value, which is the weight of validators that must exhibit Byzantine behavior (in a way that it is detected) in order to produce contradictory messages

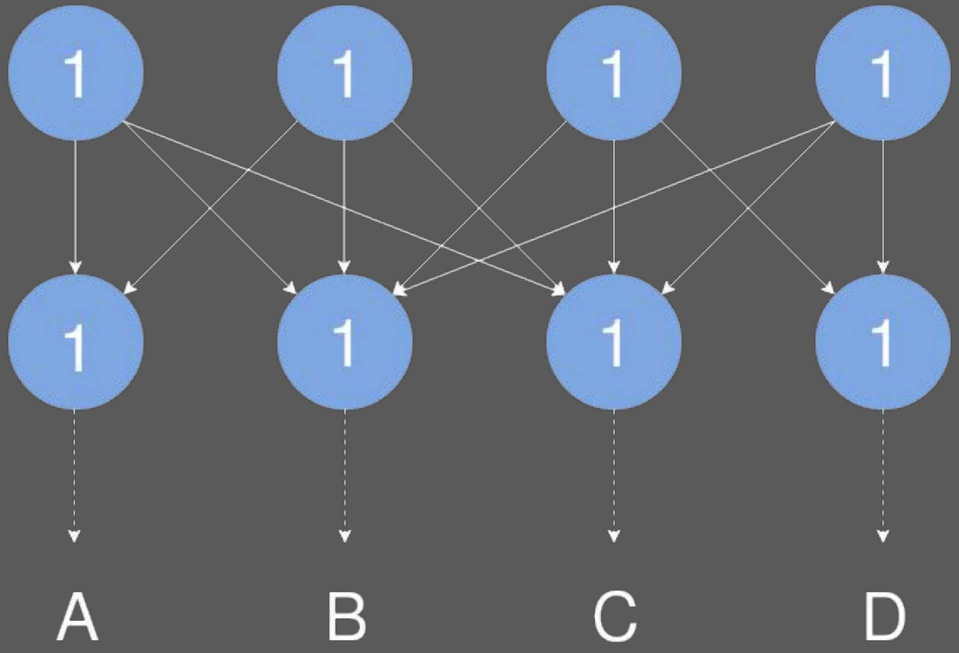
Note that the LMD rule is vital for this!

Finality Inspector



Clients can decide their own safety thresholds, and then declare a value as safe when the finality score is at least the safety threshold

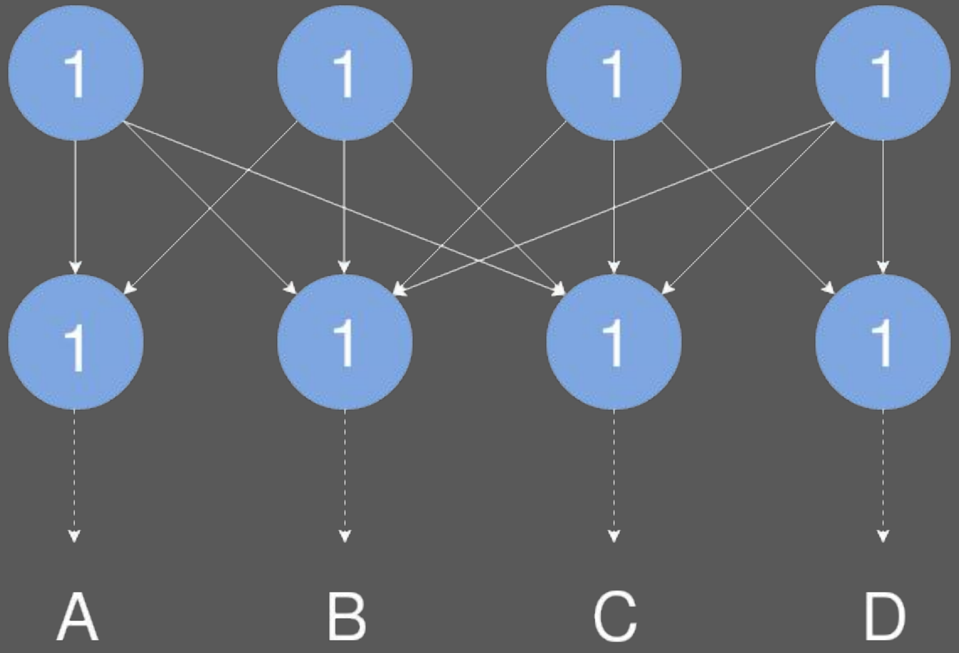
Finality Inspector



This message passing structure is known to be sufficient for some level of finality on the value “1”

The structure is similar to a q-degenerate graph

Finality Inspector



We came up with a way to efficiently extend these “witnesses” for finality on a specific value

Upon seeing a new message, the algorithm tries to extend the previous k-layer witness in O(V*log(V)*log(K))

Implementations

- github.com/CarlBeek/CBC_LMD
  - Python
  - Fast Casper CBC LMD

- github.com/sigp/lighthouse
  - Rust
  - Production-targeting Eth2.0 implementation

Ethereum 2.0 Implications

- Potentially a useful algorithm for Eth2 fork choice.
- Existing Eth2 fork-choice optimizations can also apply to this (e.g., “clear winner”).

- Needs some “extra bits” to work nicely with Eth2.
  - Blocks are not latest messages in Eth2, only attestations.
  - Naive reduced-tree does not select head in absence of attestations.

Thank you!

More CBC: github.com/cbc-casper