# Heterogeneous mempool DAGs

Typhon Team Heliax

December 1, 2022

## Contents



# 1 Introduction

**Motivation**[A] Bridges ~~suck~~ have been the source of sorrow in the past. Even if they do work as intended, at best, they connect a single pair of chains. Building bridges between every pair of chains is expensive . We want to enable any number of participants with assets on several different ledgers[B] to "interact" "directly" .

**Probblem statement**[C] `How to enable any number of actors to interact directly with moderate ressource consumption and suitable latency, using only a "minimal" number validators on the involved base ledgers.`
The ideal solution would be the atomic execution of a single transaction that makes reference to all invovled base ledgers.[1] Can we operate a protocol that

[1] This assumes counterparty discovery before the transaction is crafted.

strikes a good compromise between the number of validators necessary to order (and execute) transactions, not spending more ressources than complete pairwise bridging?

### 1.1 Overview

Roughly, we have two complementary protocols running concurrently:

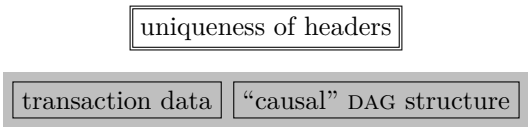



Figure 1: Illustration of the interdependence of the availability and integrity protocols

1. the availability protocol; and
2. the integrity protocol.

The availability protocol makes sure that transaction data is made available as long as necessary. The integrity protocol makes sure that each validator can only produce one block in each of its (local) rounds. However, the integrity protocol produces additional data, besides transactions, which leads to additional data to be taken care of by the availability protocol.

## 2 Architecture and communication patterns

We incorporate Narwhal's [DKSS22] scale out architecture: each validator has a unique *primary* and a number of *workers* (see Fig. 2).

## 3 Preliminaries

Let us fix an arbitrary learner graph.

**Definition 1** (Global Weak Quorum)**.** A *global weak quorum* is a set $X$ that is a weak quorum for each learner, i.e., $X \cap q_a \neq \varnothing$ for every learner $a \in L$, and all $q_a \in Q_a$.

**Definition 2** (Universal Quorum)**.** A universal quorum is a set that contains a qourum for each learner, *i.e.*, a set $q$ such that $q \in Q_a$ for all $a \in L$.

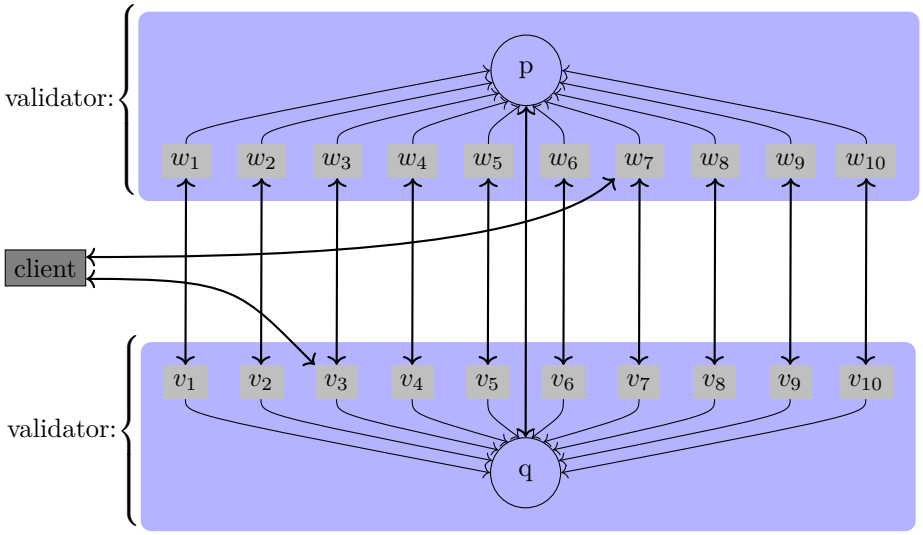



Figure 2: The structure and communication patterns of validators

# 4 Worker actions

Every validator has the same number of workers. Thus, each worker can be assigned a unique *mirror worker* on every other validator. We shall adopt the convention that mirror workers of each other share the same subscript, *e.g.*, in Fig. 2, workers $w_2$ and $v_2$ are mirror workers of each other. Workers are only tasked with "bookkeeping" matters: they keep track of (batches of) transactions, their hashes, and erasure coding shares; they only pass on hashed data and header information to their primaries. Primaries never send messages back to their workers.

## 4.1 The pure availability protocol: worker actions

**Transaction collection (TX←)** Each worker keeps listening for new transaction requests from clients.[2] Transactions should be buffered using reasonably fast memory. worker ← client

**Transaction distribution (TX⇒)** Each worker "broadcasts" erasure coding shares of every received transaction to mirror workers. In the simplest case—the one we cover first—this amounts to broadcasting the transaction.[3] In general, transaction distribution can be divided into two steps. worker ⇒ worker

[2]The bandwith and amount of storage for storing incoming transactions *should* be big enough to process all incoming transactions. We share this assumption with Byzantine set consensus [CNG21]. Transaction fees are one way to avoid flooding attacks, making the latter prohibitively expensive.

[3]However, if we perform proper erasure coding, it is not broadcasting in the strict sense: each mirror worker receives a different message. We first cover the case of trivial erasure

**Erasure coding via copying (TX)** We first cover the case of trivial erasure coding, in line with the description of the homogeneous case [DKSS22]. Thus, erasure coding shares of a transactions are simply copies. We visually distinguish between "copies" of transactions from the "original" transaction supplied by the client, using the symbols **TX** and TX, respectively. [worker]

**Copy distribution (TX⇒)** Each share of the erasure code, *i.e.*, a copy of the transaction, is sent to every mirror worker.[D] We tag each transaction with a *sequence number*, consisting of the validator round and the position in the list of transactions for the next block (header) in which the transaction will be included. worker ⇒ worker

Note that copies of transactions are *not* signed by the worker. Signatures are deferred to until after the last transaction of a validator round, when the worker will sign the hash of the list of all broadcast transactions, called a *worker hash*.

**Worker hash compilation (WH)** Towards the end of a "validator round",[E] each worker produces its worker hash for the broadcast batch of transactions. In detail, a worker hash consists of (worker)

- the hash of the broadcast list of transactions,
- the number of transactions, and
- the round number,

signed by the worker. [F]

**Worker hash broadcast (WH⇒)** A worker broadcasts its most recent worker hash to mirror workers. worker ⇒ worker

**Worker hash provision (WH↑)** The most recent worker hash is sent to the primary for inclusion into the next header. worker → primary

**Worker hash reception and checking (WH←)** At any time, a worker can receive a worker hash from a mirror worker. As a first reaction, it checks whether enough transactions have been received by the worker and, if so, whether the hash of the list of transaction matches the worker hash. worker ← worker

**Worker hash forward (WH⇑)** If a worker has successfully checked the availability of the transactions of a received worker hash, it "forwards" the worker hash to its primary.[4] worker → primary

coding covering the case of proper erasure codining in ?! .

[4] Validators will use this information to send availability commitments to block headers of other primaries.

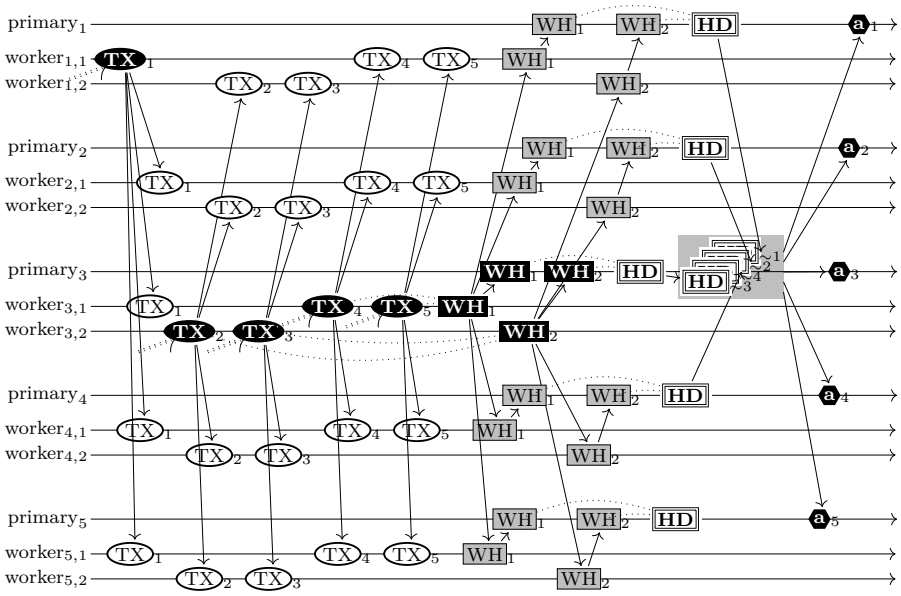



Figure 3: The availability protocol in the genesis round

# 5 Primary actions

## 5.1 Availability at genesis

The pure availability protocol: genesis actions

**Genesis header compilation (HD)** If a primary has obtained a complete set of worker hashes for the genesis round from its workers, it can compile a block header. The process of header compilation does not distinguish between worker hashes of its local workers and those with transactions that were received at other validators. At genesis, the header consists of the creator's identity and the list of worker hashes. [primary]

**Availability voting/commitment ($\boxed{\text{HD}}_p\rightarrow$)** A genesis header of a primary is acceptable if all its worker hashes have been forwarded by the local workers (which are trusted to have checked these worker hashes). The latter implies that the relevant erasure coding shares are kept available. An availability commitment is made by signing the header and sending the signed header to the creator (for the purpose of aggregation into availability certificates). primary → primary

**Commitment aggregation ($\boxed{\text{HD}}_p\leftarrow$)** The signatures of received availability commitments are aggregated into certificates of availability for a header. [primary]

**Certificate broadcast ($\text{a}\Rightarrow$)** Once a primary receives commitments from a global weak quorum for its genesis header, it broadcasts the certificate of primary ⇒ primary

availability.

**optional header distribution** One might expect that header creators send their headers around. However, there is no need for this.[G]

A (partial) execution of the availability protocol at genesis is illustrated in Fig. 3.

## 5.2 Integrity: the general case at once

First off, the integrity-protocol re-uses the sending of signed headers $\boxed{\mathbf{HD}}_p$ to the creator (from the availability-protocol), as a commitment of the signer to one unique header for the validator (and round), namely the first one signed and sent. Thus, correct validators will not sign and send any other header for the respective creator of the header (for the same round).

**Integrity signing $\boxed{\mathbf{HD}}_p$** Signing and sending the header to the creator implies that (a correct) primary will not sign any other header of the same creator with the same round number. primary ⇒ primary

**Block aggregation (bk)** Using the same signature aggregation mechanism that is used for availability certificates, validators will aggregate additional signatures (besides the availability signature) for their block headers, producing learner-specific *blocks*, which, by definition, are headers signed by a learner-specific quorum. [primary]

**Block broadcast** bk The aggregated signatures of a block header form a *learner-specific block*. The signature aggreator will broadcast to all primaries that belong to some quorum of the respective learner. (Later, these will be used as references to previous blocks in the learner-specific DAGs see **??**) primary ⇒ primary

There is no conceptual difference between the integrity protocol at genesis, comparted to the typical case. The only difference is that headers in the general case will carry additional information. Thus, we can finish the description of the protocol, by filling in the additional data and steps in the typical phase of the availability-protocol (see also Figure 5, for the difference between headers at genesis and the typical phase).

## 5.3 Availability: the typical case

**Generating and broadcasting signed quorums** Once a validator has collected enough new blocks (for a learner), it signs a learner-specific quorum of such blocks; the result is called a *signed quorum*, for short. All these blocks have to be from the same round. important primary ⇒ primary

Under certain conditions, in particular if there is exceptional delay for a specific learner, one can forgoe announcing a proper signed quorum and

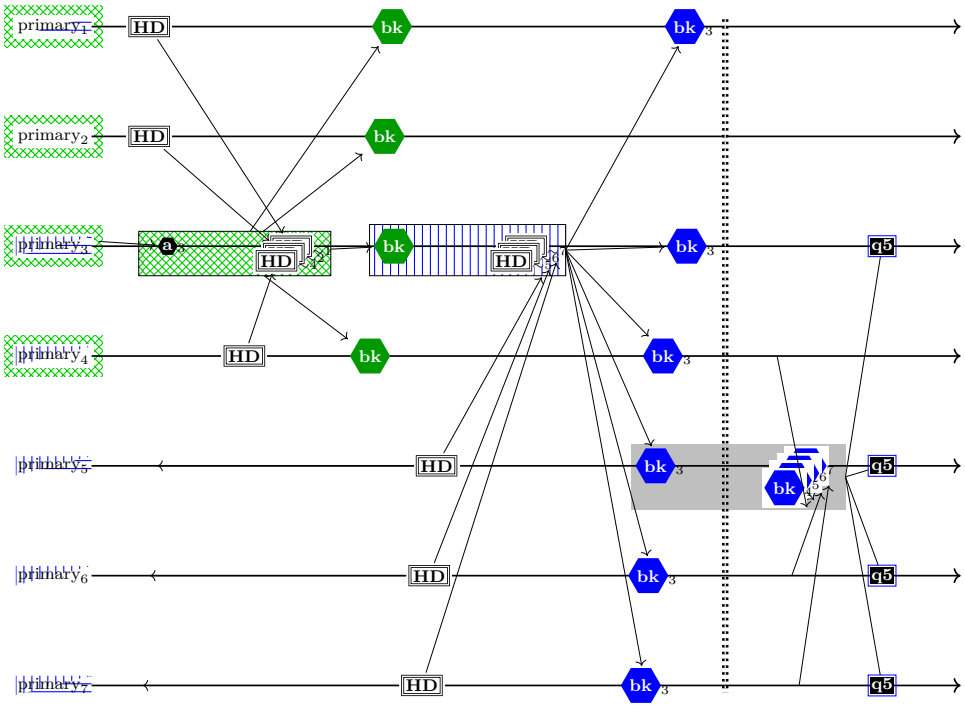



Figure 4: The integrity protocol (concluding each round that's was "opened" in the availability-protocol)

instead signs a *dummy quorum* for a specific learner, *i.e.*, a signature over the ID of the learner in question and the current round number.

**General header compilation** The biggest additional work and data concerns the compilation of headers. In the typical phase, a header carries two additional data items, namely

- the availability certificate of the previous header of the header's creator/initiator
- hashes of the signed (dummy) quorums sent by the same validator

**General header checking** As signed quorums also serve as certificates of availability, checking a signed quorum amounts to checking the signed certificates.[H] In a similar way, the certificate of availability amounts to a checking of signatures.

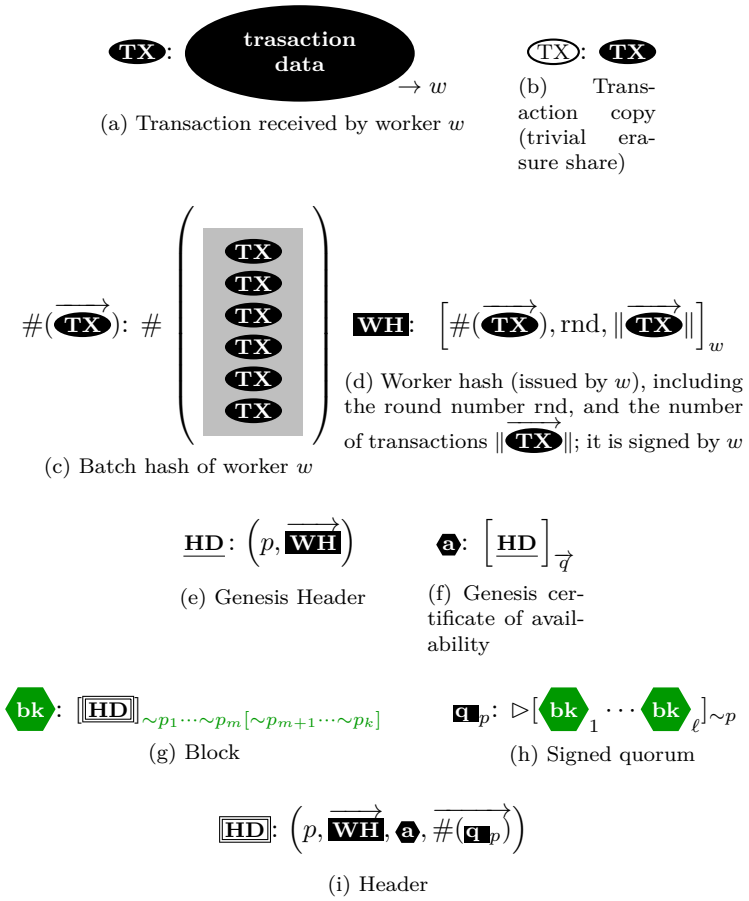
## 5.4 Summary

The availability protocol in a non-genesis round only differs in having

1. the additional requirement that each block header also includes the certificate of availability for the previous header of the same validator and
2. the sending and checking of signed quorums (each of which implements the reference to blocks from the previous roundin a learner-specific DAG).

As a consequence, casting an availability vote / sending a commitment message becomes a recursive commitment to storing all blocks until genesis (or the last block that some of learners might still want availabl).

# 6 Data structures




(a) Transaction received by worker $w$

(b) Transaction copy (trivial erasure share)

(c) Batch hash of worker $w$

(d) Worker hash (issued by $w$), including the round number rnd, and the number of transactions $\|\overrightarrow{\text{TX}}\|$; it is signed by $w$

(e) Genesis Header

(f) Genesis certificate of availability

(g) Block

(h) Signed quorum

(i) Header

Figure 5: Overview of data structures

# References

[CNG21] Tyler Crain, Christopher Natoli, and Vincent Gramoli. Red belly: A secure, fair and scalable open blockchain. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 466–483, 2021.

[DKSS22] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: a dag-based mempool and efficient BFT consensus. In Yérom-David Bromberg, Anne-Marie Kermarrec, and Christos Kozyrakis, editors, *EuroSys '22: Seventeenth European Conference on Computer Systems, Rennes, France, April 5 - 8, 2022*, pages 34–50. ACM, 2022.