Heterogenizing DAG-based consensus ($1/n$): heterogeneous narwhal-rider

Typhon Team Heliax

May 16, 2023

Abstract

Blockchains exhibit *linear* structure as each block has a unique reference to *the* previous block. If instead, each block may reference *several* previous blocks, we can build a *directed acyclic graph* (DAG) of blocks.[1] In fact, such "block DAGs" are the basic data structure that a whole family of consensus protocols relies on, namely DAG-rider, Bullshark, and Narwhal&Tusk.[2] These protocols build a growing "global" DAG of transaction data such that—among other things—(1) every validator's local view is a sub-DAG of *the* "global" DAG and (2) every block of the "global" DAG is *logged* for inclusion into a total order of blocks; the latter typically implies eventual execution of the respective transactions, *i.e.*, we aim for inclusion fairness.

The paper applies the general idea of heterogenization to the Narwhal mempool; we dub the resulting protocol *Heterogeneous Narhwhal-Rider*, or HNR for short. Like Heterogeneous Paxos is a heterogeneous version of Lamport's Paxos, HNR is a heterogeneous version of the Narwhal mempool, extended with a variation of DAG-rider's weak links—whence the name. We conclude with a discussion of how heterogenization is a suitable tool for building essentially any eco-system on the spectrum that interpolates between Vitalik's cross-chain and multi-chain worlds.

Contents

[1] Note that this includes blockchains as one particular case.

[2] The respective references for these papers are, Keidar et al. 2021, Spiegelman et al. 2022, and Danezis et al. 2022.

1 Introduction

Directed acyclic graphs feature in several recently proposed consensus protocols Danezis et al. 2022; Keidar et al. 2021; Spiegelman et al. 2022. So, do we want to add yet another DAG-based consensus protocol?

The authors' answer is a crisp *No!* The research field being extremely active, chances are somebody is in the very process of writing a new DAG-based consensus protocol. The present paper is rather about *heterogenizing* existing protocols, choosing DAG-based consensus as an example that goes together with heterogeneous Paxos I. Sheff et al. 2021.[3] Roughly, we use a mix of Narwhal and DAG-rider and we might want to think of this as an experiment that gathers evidence for how the general principle of heterogenization.

Along the way, we shall find out that heterogenization applies differently to questions of availability and integrity I. C. Sheff et al. 2019:

availability all relevant transaction data is promised to be available

integrity the protocol is safe, *e.g.*, no double spends, equivocation, *etc.*.

2 Preliminaries

We now describe the central concepts that we rely on in our description of the HNR protocol and provide formal definitions. Moreover, we also introduce terminology that will allow us to describe HNR's salient properties precisely.

We fix a finite set of *learners* $\mathbb{L}$.

[3] We are confident that heterogenization applies to the whole family of DAG-based mempools.

2.1 Opinions about quorums: learner-specific and global

The HNR protocol is designed to take into account learner-specific assumptions, first and foremost about liveness of sets of validators.[4] Each learner holds certain beliefs according to which a given behavior of a validator is deemed possible or not. Each learner is asked to commit to a set of *quorums* such that at least one of the quorums will be live at all times. A typical example for a learner-specific set of quorums is the collection of all sets of validators that are staking more than $^2/_3$ of all stake of the respective base ledger.

In general, the HNR protocol description takes is parameterized by a function from learners to sets of learner-specific quorums. Let us mention once more, as protocol designers, we assume each learner to have commited to a set of quorums, such that one of these quorums is live at all times (and that this is compatible with the learner's beliefs).[5]

Definition 1 (Learner-specific quorum system)**.** A *learner-specific quorum system* is a function from learners to sets of quorums. For a learner-specific quorum system Q and a learner a, we denote the learner's set of qourums by Q_a, whose elements are the *learner-specific quorums* of a. We use q_a, q'_a, *etc.* to range over learner-specific quorums of learner a.

One might even go as far and ask learners to choose a maximal set of quorums that is compatible with their beliefs; however, for the operational aspects of the protocol, all we need is a map from learners to sets of quorums. Now, with the definition of quorums systems at hand, we can formulate the second core definition.

Definition 2 (Global weak quorum)**.** A *global weak quorum* is a set of validators X that is a weak quorum for each learner, i.e., for every learner $a \in \mathbb{L}$ and all learner-specific quorums $q_a \in Q_a$, we have a non-empty intersection $X \cap q_a \neq \varnothing$.

Global weak quorums are important for matters of availability: if a global weak quorum is holding a copy of a particular transaction request, then one copy should be available (as a "shared belief" of all learners). The archetypal example for global weak quorums are sets of validators that (just) hold more than $^1/_3$ of the stake of *every* base ledger.

Finally, we make the simplifying assumption that learners agree with us on the following two points:

1. there is a common set of acceptors $\mathbb{A}$, obtained as the union of the (unions of the) learner-specific quorums that
2. we can assume that $q_a \in Q_a$ and $x \in \mathbb{A}$ imply $q_a \cup \{x\} \in Q_a$, w.l.o.g.

For the second point, the reasoning is that if the larger set $q_a \cup \{x\}$ is live, then in particular the "original" quorum q_a must be live.

[4] The terminology about trust assumptions was introduced already for Heterogeneous Paxos I. Sheff et al. 2021.

[5] This is formalized as `TrustLive` in the TLA$^+$-specification.

2.2 Opinions about safety: default or fully parametric

We now turn to assumptions about safety that complement the assumptions about liveness of quorum systems. For this, we would like learners to commit to assumptions about absence of Byzantine behavior and/or incentives for correct behavior. This will allow us to state conditions under which DAG properties of the Narwhal protocol will transfer to HNR. Roughly, we are looking for collections of validator sets that are "big enough" and consists of "sufficiently relevant" validators such that learners would expect each of these validators sets adequate for establishing consistency between pairs of base ledgers.

If we want to be more precise, we have to take into account that consistency is relative to a specific pair of base ledgers. Moreover, different learners may have different opinions concerning the trustworthiness of a specific validator (set) or the incentives for working towards consistency of a particular pair of base ledgers. However, before we delve into more details of which kinds of opinions about what in relation to all pairs of base ledgers we are interested in, let us point out the simple default assumption that we can derive from quorum systems: a *default safe set* for consistency of two base ledgers $a, b \in \mathbb{L}$ is a set of validators $s \subseteq \mathbb{A}$ that satisfies

$$s \cap q_a \cap q_b \neq \varnothing \tag{1}$$

for every pair of learner specific quorums $q_a \in Q_a$ and $q_b \in Q_b$. If all these validators are correct, then they can jointly make sure that the two base ledgers stay consistent.[6] The definition of a learner graph is more general, but specializes to the case of default safe sets forming the (weighted) edges between learners.

Definition 3 (Learner graphs, validity of learner graphs)**.** A learner graph for a fixed set of learners $\mathbb{L}$ and validators $\mathbb{A}$ is a pair $(Q, \_\text{–}\_)$ of a learner-specific a quorum system Q and a function $\_\text{–}\_ \colon \mathbb{L} \times \mathbb{L} \to \wp\big(\wp(\mathbb{A})\big)$ that maps pairs of learners to sets of validators, referred to as *safe sets*, such that the following hold, for all pairs of learners $a, b \in \mathbb{L}$, for all safe sets $s \in a\text{–}b$ and all validators $x \in \mathbb{A}$:

1. $a\text{–}b = b\text{–}a$, *i.e.*, edges are undirected;
2. $s \cup \{x\} \in a\text{–}b$, *i.e.*, edge weights are upward closed.

A learner graph is *valid* if for every $q_a \in Q_a$, for every $q_b \in Q_b$, $s \cap q_a \cap q_b \neq \varnothing$. A *safe set* of a learner graph is an element of $a\text{–}b$, for some $a, b \in \mathbb{L}$, *i.e.*, an element of the edge weights.

Thus a learner graph is an undirected simple graph whose nodes are learners and whose edges carry sets of safe sets as weights; moreover, each node is labeled by the set of all learner-specific quorums of the respective learner.

A simple procedure for "refining" any given learner graph, *e.g.*, the default learner graph, consists in asking each learner $a \in \mathbb{L}$ to remove safe sets from edges incident to a that are "too small", *e.g.*, too small to be sure that none of

[6] In fact, such a safe set is both a weak a-quorum and a weak b-quorum.

the validators is tempted to equivocate; for a concrete example, a learner might want to strengthen the validity requirement such that every set of the shape $s \cap q_a \cap q_b$ must be holding enough stake of base ledger a. Further discussion of the learner graph is deferred to Appendix **??**.

2.3 Discussion of desirable properties of the Narwhal DAG

We shall focus on the following properties of the Narwhal DAG Danezis et al. 2022:[7]

integrity ensures that there is at most one of a certain kind (and no “forgeries”), *e.g.*, a unique block of a validator at a certain height;

availability ensures that hash-referenced data is retrievable, *e.g.*, the transaction data of a batch referenced in a worker hash;

$\frac{2}{3}$-causality ensures that each node in the Narwhal DAG has a quorum of direct causal predecessors, *e.g.*, each block has to reference three out of four blocks if four blocks have been produced.

We shall speak *wide causality* instead of $^2/_3$-causality as we are working with more general quorum systems. The above properties ensure that it is enough to choose a “chain” of anchors[8] in the Narwhal DAG where anchors are nodes have a weak quorum of causal successors.

3 Synopsis: heterogenizing the “global” DAG

Running Narwhal ibid. leads to a continuously growing “global” DAG of blocks that all validators agree on at every given point in time if enough of them are correct and alive. For the sake of simplicity, we

1. put ourselves into the seat of the omniscient observer such that it makes sense to talk about the “global” DAG (although it cannot be known) and
2. we disregard the hash referencing mechanism for transaction data (and the accompanying certificates of availability of the referenced data).

These simplifying assumptions allow for meaningful illustrations of Narwhal’s *mempool* DAG, or *mem*-DAG for short.

The two central features of protocol heterogenization are the following:

1. there is one instance of the original protocol for each learner;
2. one overarching protocol interweaves the learner-specific instances.

[7] We have omitted the chain quality rule as it seems to fall into a different category: in Anoma, there is just no incentive to introduce spurious transactions or even blocks as this would incur substantial costs with no apparent benefit.

[8] We shall speak of anchor blocks as they need not necessarily be proposed by a leader.

A simplistic way of connecting two Narwhal DAGs is illustrated in Figure 1. We simply superimpose the DAGs of the blue and red learner, requiring that the validators that hold both blue and red stake to reference both blue and red blocks. If we choose anchor blocks only from the red and blue kind all the results from Narwhal transfer to the superposition. However there is one main issues with this approach: the system is live only if the liveness assumptions of *both* learners are correct.

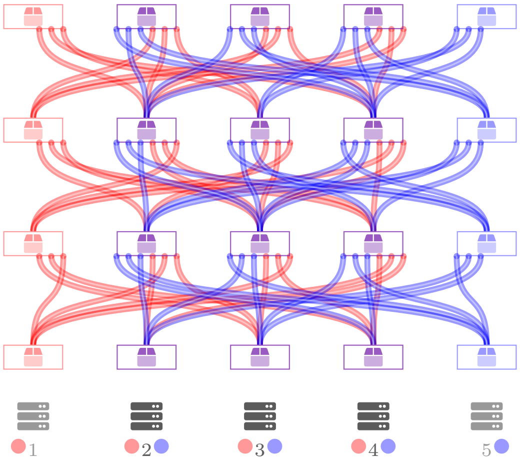


Figure 1: Superposition of two Narwhal DAGs

The challenge is that we must allow for the possibility that the overarching protocol is only running the instance of one single learner under the assumption that all other learner-specific sub-protocols will catch up. Thus, we are aiming for a maximally asynchronous collaboration of the learner-specific sub-protocols.

4 Architecture and communication patterns

Hnr incorporates Narwhal's Danezis et al. 2022 scale out architecture: each validator has a unique *primary* and a number of *workers* (see Fig. 2). 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 identifiers of mirror workers share the same subscript. For example, in Fig. 2, workers w_2 and v_2 are mirror workers of each other.

5 Worker actions (availability and executability)

Workers are in charge of availability of transaction requests of clients. They have to keep the transaction data available until execution. Specifically, workers organize transactions into batches whose hashes will be included in blocks. Availability will be achieved using a trivial erasure coding scheme, along the

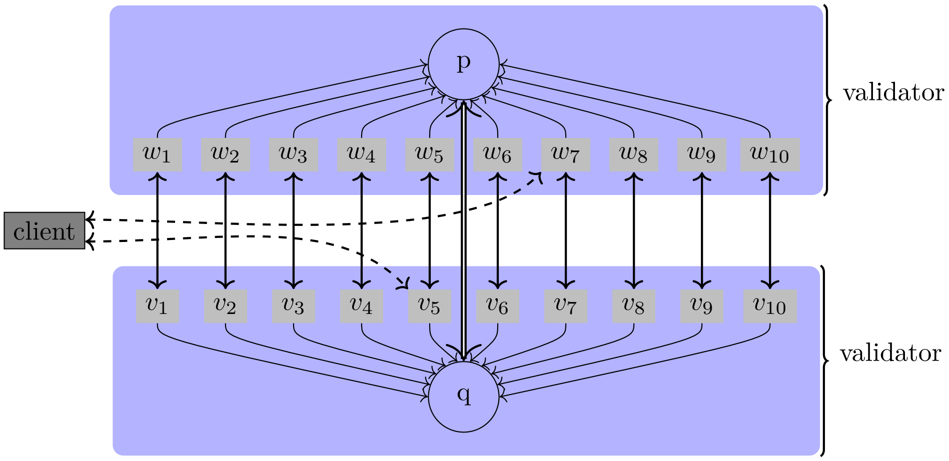


Figure 2: The structure and communication patterns of validators

lines of the Narwhal DAG Danezis et al. 2022. The core information that workers provide to their primaries are hash references to transaction requests, accompanied with consecutive batch numbers and signatures of (other) workers that commit to keeping the data available. In this way, primaries can save network bandwidth usage. Finally, as an orthogonal principle, primaries do not send messages or other direct signals to their workers.[9]

5.1 The pure availability protocol (pre-execution)

MessageEnum::TxReq **Transaction request collection (TX←)** Each worker keeps listening for incoming transaction requests from clients.[10] Transaction requests should be buffered using reasonably fast memory (to ensure that all requests are eventually served); transaction request fees may be imposed to control the rate of client requests. worker ← client

??? **Transaction batching (TXs := TXs : TX)** Every worker stores the received transaction request to *the* current batch TXs, which is list of transaction requests. This happens unconditionally, *i.e.*, there is always a unique current transaction batch and every transaction request has to be added to the current batch. New batches may be created over time, but each batch in this “stream” of batches has [worker]

[9] However, workers receive signals from executor nodes upon successful execution of transactions, which allows the workers to free up the transaction storage.

[10] 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 Crain, Natoli, and Gramoli 2021. Transaction fees are one way to avoid flooding attacks, making the latter prohibitively expensive. For example, we might consider using a FIFO-buffer; however a priority queue that takes into account a combination of fees and quality of service considerations is more suitable for managing the flow of incoming transaction requests.

a unique *batch number*. Within a batch, transactions are assigned consecutive sequence numbers: the *sequence number* of a transaction is the position in the current batch. Thus, each transaction request by identified by its *batch*, *sequence number* and the worker ID; we shall refer to this as the *fingerprint* of the transaction request.[11]

TxToAll **Transaction broadcasting (TX!TX⇒)** If a worker receives a transaction request, it will broadcast a copy to all mirror workers. In principle, we could use arbitrary erasure coding schemes. However, in line with the original version of Narwhal Danezis et al. 2022, we use full copies as erasure codes. Despite the coincidence of erasure codes and the "original" data, we visually distinguish the "copy" of a transaction from the "original" transaction supplied by the client, using two different symbols, namely TX and TX, respectively. When broadcasting copies of received transaction requests, the message also includes the fingerprint of the transaction request.

worker
⇒worker

WHxToAll,WorkerHx **Worker hash broadcast (TX!WH↑⇒)** When a worker receives a transaction request, this might trigger a new worker hash to be produced. In principle, the decision for when to send it is for the worker to decide (as long as each worker hash contains at least one transaction and does not exceed a potential maximum size).[12]

worker
⇒ worker
→ primary

WorkerHashData,WorkerHashSignature The new worker hash consists of

- the hash of the current batch TXs,
- the length of the current batch,
- the identifier of the current worker,
- a signature of the above data by the current worker.

The worker hash broadcast involves the following steps:

1. broadcasting the new worker hash to mirror workers;
2. sending the new worker hash to the worker's primary;
3. resetting the current transaction request buffer to the empty list;
4. incrementing the batch number;
5. last, but not least, storing the transactions of the current batch for retrieval.

MessageEnum::TxAck **Transaction request acknowledgment** Optionally, one may acknowledge the client's requests.

MessageEnum::TxToAll **Transaction copy (TX←)** Upon receiving a transaction copy, the worker has to store the copy locally such that its hash can be retrieved quickly via its fingerprint, *i.e.*,

worker
←worker

[11] The order of transactions within a batch might be permuted before execution to avoid potential MEV.

[12] At which exact moment worker hashes are compiled can depend on several factors, *e.g.*, on a maximum number of transaction requests per worker hash or a maximum delay between the first and the last transaction request within a worker hash.

- the ID of the collecting worker,
- the batch number of the batch to which it belongs, and
- the sequence number within the batch to which it belongs.

Note that copies of transactions are *not* signed by the worker. Signatures are deferred to worker hash broadcast. Storing transaction copies as above will be useful for handling "foreign" worker hashes, *i.e.*, worker hashes that refer to transactions that are collected at other workers.

WHxFwd **Worker hash forwarding ($\boxed{\mathrm{TX}}$!$\boxed{\mathbf{WH}}\uparrow$)** In case, the transaction was the last missing one to match a previously received worker hash, the worker hash is send to its primary. The (pre-)condition is that all transactions that are referenced in the worker hash have already been received and stored.[13] worker →primary

WHxToAll **Worker hash reception ($\boxed{\mathbf{WH}}\leftarrow$)** When a worker receives a worker hash from a mirror worker there are two cases: either it has already has received all the referenced transaction copies (and can forward the worker hash to its primary) or it has to store the worker hash (and the worker hash forwarding will be triggered by the last transaction copy). worker ← worker

MessageEnum::WHxFwd **Worker hash forwarding ($\boxed{\mathbf{WH}}$!$\boxed{\mathbf{WH}}\uparrow$)** If a worker has already stored the transactions of a received worker hash, it sends the worker hash to its primary.[14] worker → primary

We have described the protocol in terms of what workers do in reaction to *receiving* messages. This emphasizes that sending a message might be due to several, slightly different types of scenarios (depending of which message from a *set* of triggering messages arrives last and thus becomes the final trigger).

5.2 Execution support protocol

6 Primary actions

Primaries will follow a protocol in which we can distinguish between matters of availability and matters of integrity. For the availability protocol, we shall treat first the special case at genesis and later describe the additional (re-)actions in the typical mode of operation. The integrity protocol is described in between the two; after all, the two protocols are closely intertwined.[15]

[13] This might be unexpected order of events, but the sending of transaction data might be "delayed".

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

[15] Specifically, the integrity protocol produces blocks, and signed quorums of these are necessary for block headers (after genesis); however, header announcement and signing are part of the availability protocol.

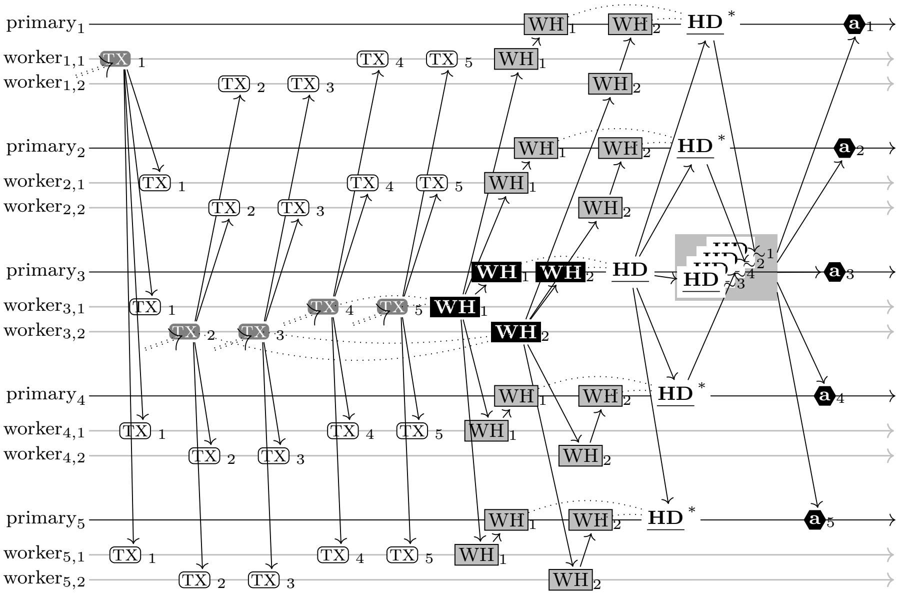


Figure 3: The availability protocol in the genesis round

In the protocol description for validators, we avoid copying text if a message has several triggers by simply listing the alternative triggers of a message (after the "default" trigger). For the pure availability protocol for workers, this would lead to a single heading for worker hash forwarding.

Worker hash forwarding ((WH/TX)!WH↑)

So, the primary trigger for worker hash forwarding are worker hashes, but transaction copies are alternative triggers (in cases of message delay irregularities).

6.1 Availability at genesis

WorkerHx **Worker hash completion (WH←)** When a worker hash (that was compiled by a local worker) is received, the primary adds the worker hash to the current list of worker hashes WHs. primary ←worker

NextHeader **Header announcement (WH! HD ⇒)** The primary may announce the next header (if the primary considers it is "time" to do so—cf. Footnote 12). The genesis header consists of the current list of worker hashes, tagged with the identifier of the primary; the round number zero is implicit. The message of the batch announcement only contains the *fingerprint* of a header, namely primary ⇒primary

- the identifier of the primary
- a list of pairs of a batch number and a worker ID.

The actual header itself consists of

- the identifier of the primary and
- the list of worker hashes.

WorkerHx **Worker hash reception** ($\boxed{\mathbf{WH}}\leftarrow$) When a worker hash that stems from a worker on a different validator is received, the primary adds the worker hash to the current list of known worker hashes.

primary ←worker

Header signature commitment ($\boxed{\mathbf{WH}}\mathbf{!}\boxed{\boxed{\mathbf{HD}}}_{\sim}\rightarrow$) If the received worker hash completes the list of known worker hashes to match a previously received header fingerprint, the primary commits to the header by signing the header and sending the signed header back to the creator of the header.

primary →primary

Header announcement ($\underline{\mathbf{HD}}^{*}\leftarrow$) If the primary receives a header fingerprint from a another primary, it is stored as long as the header might still be included into some learner-specific DAG. We also elide to mention that each correct validator has to check whether the received data are consistent with the local observations.

primary ←primary

Header signature commitment ($\underline{\mathbf{HD}}^{*}!\boxed{\boxed{\mathbf{HD}}}_{\sim}\rightarrow$) If all worker hashes of the received header fingerprint are known to the primary, the primary commits to the header by answering with a signature over the header.

primary →primary

Header signature ($\boxed{\boxed{\mathbf{HD}}}_{p}\leftarrow$) The signature of a received availability commitment is either stored or added to the aggregated signature "under construction", leading towards a certificate of availability. If the received signature completes a global quorum, it triggers the broadcasting of the completed aggregated signature, *i.e.*, the certificate of availability.

primary ←primary

Availability certificate broadcast ($\boxed{\boxed{\mathbf{HD}}}_{p}\mathbf{!}$ⓐ$\Rightarrow$) If the received commitment completes a global weak quorum for its genesis header, it broadcasts the certificate of availability.

primary ⇒primary

A (partial) execution of the availability protocol at genesis is illustrated in Fig. 3. Also, note that in some exceptional circumstances, a received header signature $\boxed{\boxed{\mathbf{HD}}}_{p}$ might also need action according to the integrity protocol, as explained next.

6.2 Integrity: the general case at once

First off, the integrity-protocol re-uses the sending of signed headers $\boxed{\boxed{\mathbf{HD}}}_{p}$ 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 from the creator of the header (for the same round).

Integrity signing *Signing and sending a header to its creator implies that (a correct) primary will not sign any other header of the same creator with the same round number. (Recall that the headers at genesis implicitly are of round zero.)*

The availability protocol for primaries will use counterparts to blocks in Narwhal Danezis et al. 2022, which come with references to a quorum of blocks, namely *learner-specific blocks* and *signed block quorums*, defined as follows: a *learner-specific block* is a block header signed by a learner-specific quorum and a *signed quorum* is a quorum of blocks signed by a primary. Finally, a *typical header*, *i.e.*, one after genesis, consists of

- the identifier of a primary
- a list of worker hashes
- an availability certificate for the previous header of the same primary
- the round number
- a non-empty list of signed quorums, at most one per learner.

The first two items were already present in genesis block headers while the remaining three only come into play in later rounds. With these definitions in place, we can describe the primary actions in the availability protocol.

Header signature ($\boxed{\mathbf{HD}}_p\leftarrow$) If the received signature of a header (interpreted as integrity commitment) completes a full learner-specific quorum of signatures, the received signature triggers the broadcast of one (or several) learner-specific blocks.

primary
←primary

> **Block broadcast ($\boxed{\mathbf{HD}}_p!(\text{bk}\Rightarrow)^+$)** If the received header signature completes a *learner-specific* block, for each such new block, the signature aggregator will broadcast a block to all primaries that belong to some quorum of the respective learner.[16] The (learner-specific) round number of a (learner-specific) block is derived from the block header that it is based on: it defaults to zero, unless there is a signed quorum that is the last one for the respective learner in the chain of block headers of the same primary and then it is one plus the signed quorum's round number.[17]

primary
⇒ primary

Note that there is no conceptual difference between the integrity protocol at genesis, compared to the typical case. The only difference is that headers need to carry additional information. Now, we can finish the description of heterogeneous Narwhal, by filling in the missing detail of the availability-protocol after genesis (see also Fig. 5, for the difference between headers at genesis and afterwards).

[16] In other words, the broadcast is to all primaries in the union of all quorums of the respective learner.

[17] Signed quorums will be used as references to previous blocks in the learner-specific DAGs, as detailed in Section/Appendix**??**.

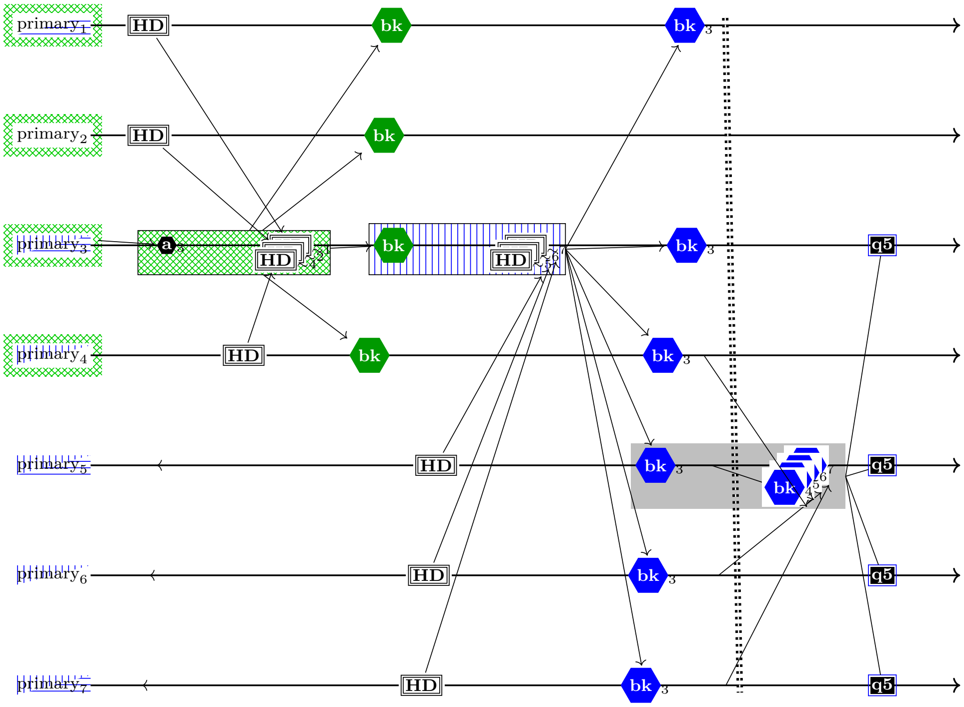


Figure 4: The integrity protocol (on the left) and the availability protocol (on the right)

6.3 Availability after genesis

The only additional message that we have to respond to are blocks. All other responses to messages are verbatim the same as at genesis.

Block reception (bk←) If a received learner-specific blocks completes a quorum of block (for the same learner) and they all carry the same round number, the primary signs and broadcasts the quorum *unless* a signed quorum of a blocks of a higher round number has been broadcast already.

primary
←primary

Signed quorum broadcast (bk!q⇒) Under the stated conditions, the primary signs a list of quorums (in the order that they were received) and broadcasts them to all primaries.

primary
⇒primary

Header announcement ((bk/WH/a)!HD*⇒) If the primary has already enough worker hashes for a new block header, it has an availability certificate of its previous header , and the produced signed quorum the first one it knows of, a new block header is announced by broadcasting its fingerprint. After genesis, a header carries two additional data items, namely

- the availability certificate of the previous header (collected by the same primary)
- a non-empty list of hashes of signed quorums sent by the same primary, but at most one per learner
- the primary's round number, obtained by incrementing the round number of the block header that this referenced in the availability certificate.

Alternative triggers are

- a (first) worker hash arriving, in which case the header announcement includes the whole list of all signed quorums (of maximal learner-specific heights)
- the availability certificate being the missing data, which then leads to all signed quorums (of maximal learner-specific heights), and as many worker hashes as possible.

6.4 Summary

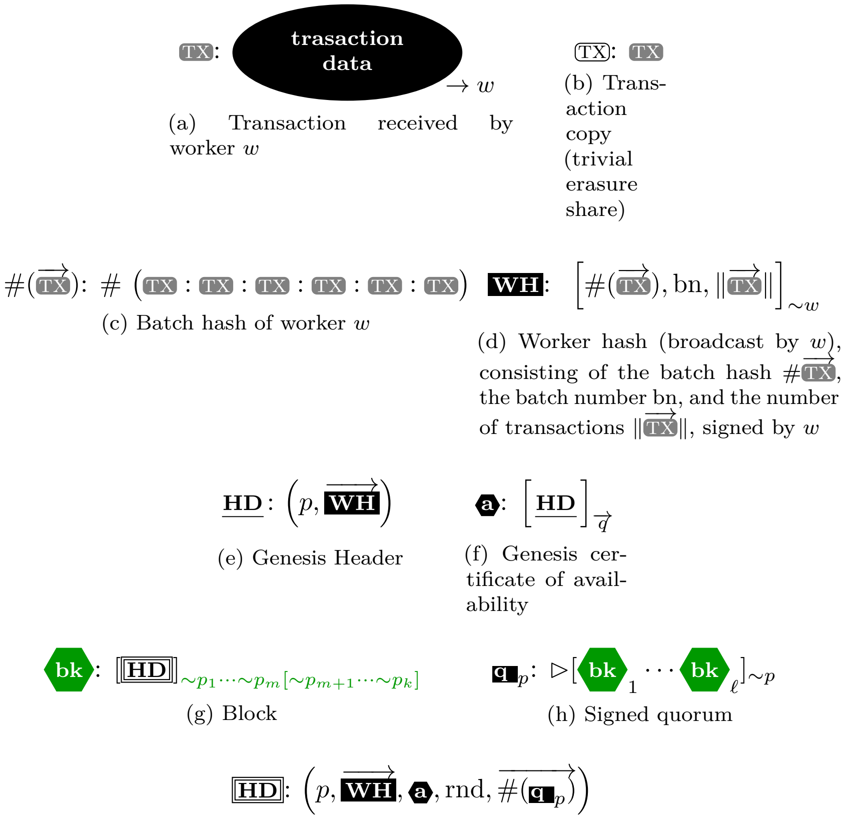
The availability protocol in a non-genesis round only differs in having

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 round—in 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 available).

7 Data structures


TX:
trasaction data
→ w
TX: TX
#(TX): # (TX : TX : TX : TX : TX : TX)
WH: [#(TX), bn, ‖TX‖]~w
HD : (p, WH)
a: [HD]q
bk: [HD]~p1···~pm[~pm+1···~pk]
q p: ▷[bk 1 ··· bk ℓ]~p
HD: (p, WH, a, rnd, #(q p))

(a) Transaction received by worker w

(b) Transaction copy (trivial erasure share)

(c) Batch hash of worker w

(d) Worker hash (broadcast by w), consisting of the batch hash $\#\overrightarrow{\text{TX}}$, the batch number bn, and the number of transactions $\|\overrightarrow{\text{TX}}\|$, signed by w

(e) Genesis Header

(f) Genesis certificate of availability

(g) Block

(h) Signed quorum

(i) Header

Figure 5: Overview of data structures

8 Transaction life cycle

A transaction goes through the following stages, first on the worker, then on the primary, finally off to execution. In more detail, on a "sunny day", the progress towards execution, step by step, is as follows:

- submission to a worker, received as TX ←
- the transaction gets assembled into a batch that is referenced by a worker hash, which in turn is sent to the primary and received as WH
- the worker hash is included into a block header $\boxed{\mathbf{HD}}$
- the block header is announced, received as $\boxed{\mathbf{HD}}^*$
- the block header is provably available, via reception of enough header signatures $\boxed{\mathbf{HD}}_p$

- the block header has reached integrity, via reception of enough header signatures $\boxed{\textbf{HD}}_p$, *i.e.*, the transaction is now referenced by a learner-specific block bk
- the block gets distributed
- the block gets a quorum of signatures q
- eventually,[18] the block gets commited,
  - either if the signed quorum makes it into a leader block,
  - or by indirect reference (see **??** for more detail)
- the block containing the transaction gets executed.

9 Comparison with Narwhal

Let us see how the special case of a single learner amounts to a slightly more complicated version of Narwhal. For workers nothing really chances. Note that due to the restriction that a new block header has to contain at least one signed quorum (and at most one), this means that every new header must contain a signed quorum; moreover, the signed quorum is a quorum of signatures for a previous block of the unique learner.

10 The chain of anchor blocks

A block can be proposed as anchor block if the following conditions are all met:

non-triviality the underlying header must include at least one signed quorum (unless the block is at genesis);

succession the previously chosen anchor block must be reachable via references in the form of signed quorums or availability certificates (unless the block is at genesis);

connectivity for each signed quorum of the underlying header, the anchor block is referenced by a weak quorum of the respective learner by direct causal successors;

readiness there is no pending consensus.

[18] In fact, this is the crucial point (see also **??**).

On pending consensus Consensus on anchor blocks is via Heterogeneous Paxos, abbreviated as P*. Note that P* may reach consensus relative to each learner in such a way that as soon one of the learners has reached consensus all connected learners will eventually have to come to the same consensus (albeit "delayed", in theory indefinitely). To avoid global delays, new proposals for anchors are possible if, for every signed quorum of the proposed block, the respective learner has reached consensus for the previous anchor block (unless the block is at genesis).

Inclusion fairness Typically, there are reward systems such that, as long as new blocks are added to the DAG, the rewards for committing an anchor block for a given learner will grow as the "backlog" of block headers for the respective learner grow such that eventually incentives will grow to add anchor blocks for the respective learner.

References

Crain, Tyler, Christopher Natoli, and Vincent Gramoli (2021). "Red Belly: A Secure, Fair and Scalable Open Blockchain". In: *2021 IEEE Symposium on Security and Privacy (SP)*, pp. 466–483. DOI: `10.1109/SP40001.2021.00087`.

Danezis, George et al. (2022). "Narwhal and Tusk: a DAG-based mempool and efficient BFT consensus". In: *EuroSys '22: Seventeenth European Conference on Computer Systems, Rennes, France, April 5 - 8, 2022*. Ed. by Yérom-David Bromberg, Anne-Marie Kermarrec, and Christos Kozyrakis. ACM, pp. 34–50. DOI: `10.1145/3492321.3519594`. URL: `https://doi.org/10.1145/3492321.3519594`.

Keidar, Idit et al. (2021). "All You Need is DAG". In: *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*. PODC'21. Virtual Event, Italy: Association for Computing Machinery, pp. 165–175. ISBN: 9781450385480. DOI: `10.1145/3465084.3467905`. URL: `https://doi.org/10.1145/3465084.3467905`.

Sheff, Isaac et al. (2021). "Heterogeneous Paxos". In: *24th International Conference on Principles of Distributed Systems (OPODIS 2020)*. Ed. by Quentin Bramas, Rotem Oshman, and Paolo Romano. Vol. 184. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 5:1–5:17. ISBN: 978-3-95977-176-4. DOI: `10.4230/LIPIcs.OPODIS.2020.5`. URL: `https://drops.dagstuhl.de/opus/volltexte/2021/13490`.

Sheff, Isaac C. et al. (2019). "Charlotte: Composable Authenticated Distributed Data Structures, Technical Report". In: *CoRR* abs/1905.03888. arXiv: `1905.03888`. URL: `http://arxiv.org/abs/1905.03888`.

Spiegelman, Alexander et al. (2022). "Bullshark: DAG BFT Protocols Made Practical". In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. CCS '22. Los Angeles, CA, USA: Association for Computing Machinery, pp. 2705–2718. ISBN: 9781450394505. DOI: `10.`

1145/3548606.3559361. URL: https://doi.org/10.1145/3548606.3559361.