Heterogeneous Narwhal: a cross-chain mempool DAG

Typhon Team Heliax

January 3, 2023

Abstract

Blockchains exhibit *linear* structure, resulting from a *single* reference to the previous block. If instead, each block may reference *several* previous blocks, we can build a *directed acyclic graph* (DAG) of blocks. In fact, such "block DAGs" are the basic data structure that several recent consensus algorithms rely on, *e.g.*, DAG-rider, Bullshark, and Narwhal&Tusk.[1] 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 node of the "global" DAG is "logged" for inclusion into a total order of blocks. Such "global" DAGs will be the topic of the present paper, referred to as Mempool DAGs, or *mem-DAGs*, for short.

The paper introduces a cross-ledger mem-DAG, generalizing Narwhal's single ledger mem-DAG; in analogy to Heterogeneous Paxos, we dub it *Heterogenous Narwhal.* Heterogeneous Narwhal&Paxos are an alternative for bridges. We also describe how being scheduled for inclusion should lead to eventual inclusion, and which hurdles need to be taken to achieve this.

we actually need to explain how heterogeneous Narwhal is 1. an alternative to bridges 2. which benefits / drawback we have, in general 3. which parts are "implementation detail".

Contents

[1] The respective references are,

1 Introduction

main contribs:

- alternative to bridges (between very different chains ?)
- merging might be actually a very good thing (for very small chains)
- chimera chains (between big chains)

In summary, we present a general framwork for building a cross-chain ecosystem of pre-existing and new ledgers.

2 Motivation / Background / Context

2.1 Atomic cross-chain swaps

> An atomic cross-chain swap is a distributed coordination task where multiple parties exchange assets across multiple blockchains, for example, trading bitcoin for ether.
>
> An atomic swap protocol guarantees (1) if all parties conform to the protocol, then all swaps take place, (2) if some coalition deviates from the protocol, then no conforming party ends up worse off, and (3) no coalition has an incentive to deviate from the protocol. [Her18]

2.2 Let's start in the land of naïveté

2.2.1 The impossible

One simple "ideal" solution for cross-chain swaps is "concurrent" inclusion of transactions in two blocks, one on each chain, which refer to each other reciprocally. However, this simply cannot be accomplished with hash links.

2.2.2 A seemingly good situation

Another naïve approach consists in is running a validator on both involved chains. Whenever the validator has the right to propose the next block on both chains, we "simultaneously" propose one transaction on each chain. However, we could "loose" the proposal spot, *e.g.*, if a view change is initiated on one of the two chains. Even worse, it could be that one of the proposals goes through, while the other gets "vetoed".

describe in more detail

2.2.3 The envisioned solution: chimera chains

We propose to establish a ledger that records transactions that involve two (or more) pre-existing chains—dubbed *chimera chain*. If we want to speak in terms of bridges, our envisioned solution combines the ideas of trusted and trustless bridges.

add fine print on which bridges might actually do this already: in particular, trustless bridges, see *e.g.*, `https://ethereum.org/en/bridges/`

The chimera chains orders blocks with cross-chain transactions such that inclusion in the chimera chain leads to (eventual) execution in the two base ledgers . The seemingly simple principle for running the chimera chain is maximal usage of pre-existing validator sets. In particular, we want

1. more than a single validator in the intersection of two quorums
2. a protocol, that gives these validators the opportunity to propose *combo block* on a chimera chain.

explain in non-technical terms that the same-slot-on-both-chains situation has a very intuitive counterpart in having quorum-pairs between two chains as the quorum system for the chimera chain of two chains.

2.3 Chimera chains

Assuming a fixed set of base ledgers , each of which have a set of quorums such that one of them "should" be live at any point in time,[2] chimera chains have the following phases:

explain this

Spawn Among other things, the genesis blocks are generated, including links to the blocks in all base ledgers that "summoned" the chimera chain.

Active The usual mode of operation, in particular facilitating cross-chain transfers.

Retire Low activity or inconsistencies might call for retirement of the chimera chain.

Merge In principle, the ecosystems of two base ledgers do "completely" merge such that the chimera chain is still performant enough. In fact, this might be a good thing: smaller base ledgers are just joining larger ones. Clearly, there's a trade off, between centralization and usability.

The chimera chain has dedicated "administrative" transactions, e.g., for spawing

3 Overview: mem-DAGs, heterogeneity, etc.

Recall that mem-DAGs build a DAG of blocks, each referencing a quorum of previous blocks; a block in the DAG can be *committed* (using virtually any consensus algorithm) if it is referenced by a weak quorum (from the next "layer" in the DAG). An example of such a DAG is shown in Fig. 1. In the present paper, we describe a protocol for concurrent building of entangled DAGs for a whole ecosystem of different ledgers, in the spirit of a cross-chain world.

Roughly, we have two complementary protocols running concurrently:

[2]This is a simplifying assumption, which we use only in the exposition. In principle, learners are completely independent entities. They "dictate" which quorums they would trust, in particular concerning liveness. For safety, the situation is more intricate.

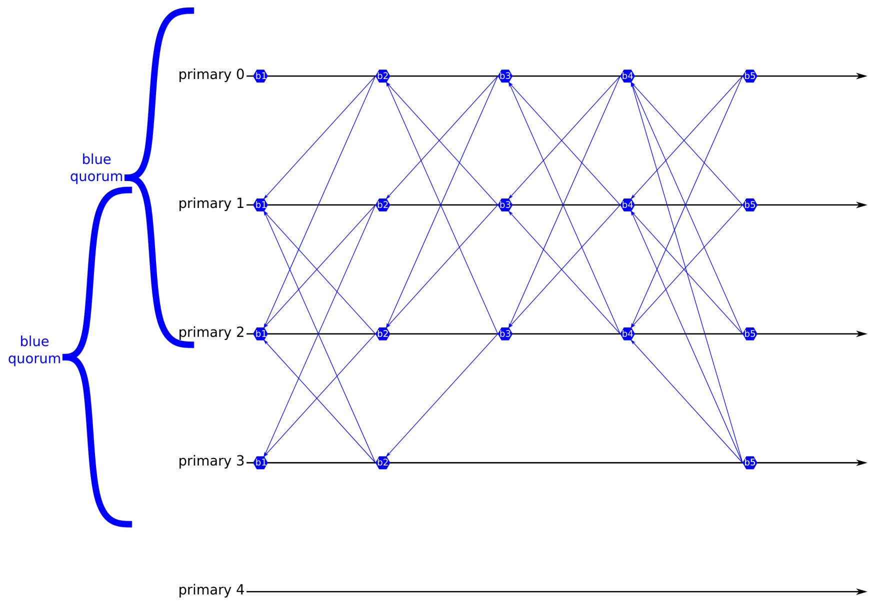
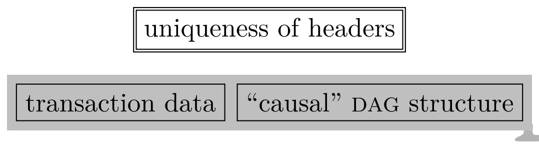


Figure 1: A mem-DAG (self-references omitted)



where to put the signed quorums ?!

Figure 2: 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 available as long as necessary;[3] moreover, the availability protocol is tasked with keeping available the *signed headers*,

explain `signed headers`

which the integrity protocol produces.

The integrity protocol makes sure that each validator can only produce one block in each of its (local) rounds.

[3]There is some fine print concerning the conditions under which this is actually the case.

4 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. 3).

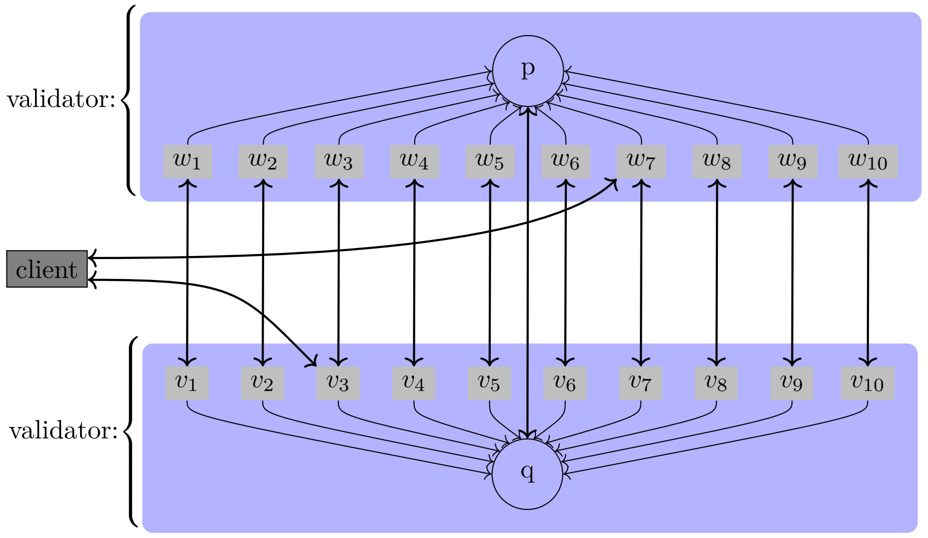


Figure 3: The structure and communication patterns of validators

5 Preliminaries

"copy" relevant parts of the heterogeneous paxos tech report

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.

"upward" closure for live quorums seems a "wrong" assumption

6 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 of using the same subscript for mirror workers of each other. Thus, in Fig. 3, workers w_2 and v_2 are mirror workers of each other. In the integrity protocol, workers are only tasked with "bookkeeping" matters; in particular, they keep track of (batches of) transactions, their hashes, and erasure coding shares; they only pass on hashed data and block header information to their primaries. The idea is to keep the network bandwidth usage of primaries as low as possible. For example, primaries do not need to send messages back to their workers.

6.1 The pure availability protocol: worker actions

get rid of the broadcast of the availability certificates

maybe we need to "announce" headers, to avoid "crossing" of round numbers of learner

so, signed qourums are an "output" of the integrity protocol, and need to be made available (such that it is possible to create new headers).

worker hashes also should include an availability certificate of the header creator's previous header

Transaction collection (TX$\leftarrow$) Each worker keeps listening for new transaction requests from clients.[4] Transactions should be buffered using reasonably fast memory.

worker $\leftarrow$ client

Transaction distribution (TX$\Rightarrow$) 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.[5] In general, transaction distribution can be divided into two steps.

worker $\Rightarrow$ worker

we might require a *position number* for transactions of the current batch, and a sequence number (of the block to be produced)

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]

put fwd ref / future work

[4]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. For example, we might assume a FIFO-buffer although it is more likely a priority queue, based on a combination of fees and quality of service considerations.

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

Copy distribution (TX⇒) Each share of the erasure code, *i.e.*, a copy of the transaction, is sent to every mirror worker. 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", 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.

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

How much "additional" information do we have to include into the header such that signing a header certificate becomes meaningful?

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.[6]

worker
→ primary

7 Primary actions

7.1 Availability at genesis

The pure availability protocol: genesis actions

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

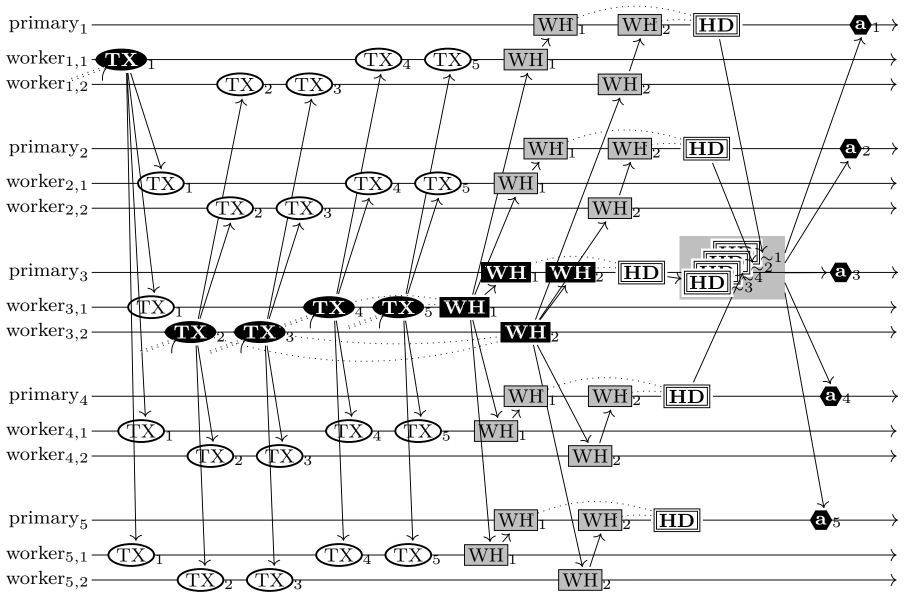


Figure 4: The availability protocol in the genesis round

Genesis header compilation ($\underline{\textbf{HD}}$) 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 ($\boxed{\textbf{HD}}_p\!\rightarrow$) 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 $\rightarrow$ primary

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

Certificate broadcast (a$\Rightarrow$) Once a primary receives commitments from a global weak quorum for its genesis header, it broadcasts the certificate of availability. primary $\Rightarrow$ primary

optional header distribution

Right now, there seems no reason to add this "safe guard"; it might be confusing.

One might expect that header creators send their headers around. However, there is no need for this.

A (partial) execution of the availability protocol at genesis is illustrated in Fig. 4.

7.2 Integrity: the general case at once

First off, the integrity-protocol re-uses the sending of signed headers $\boxed{\boxed{\textbf{HD}}}_p$ 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 $\boxed{\boxed{\textbf{HD}}}_p$ 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]

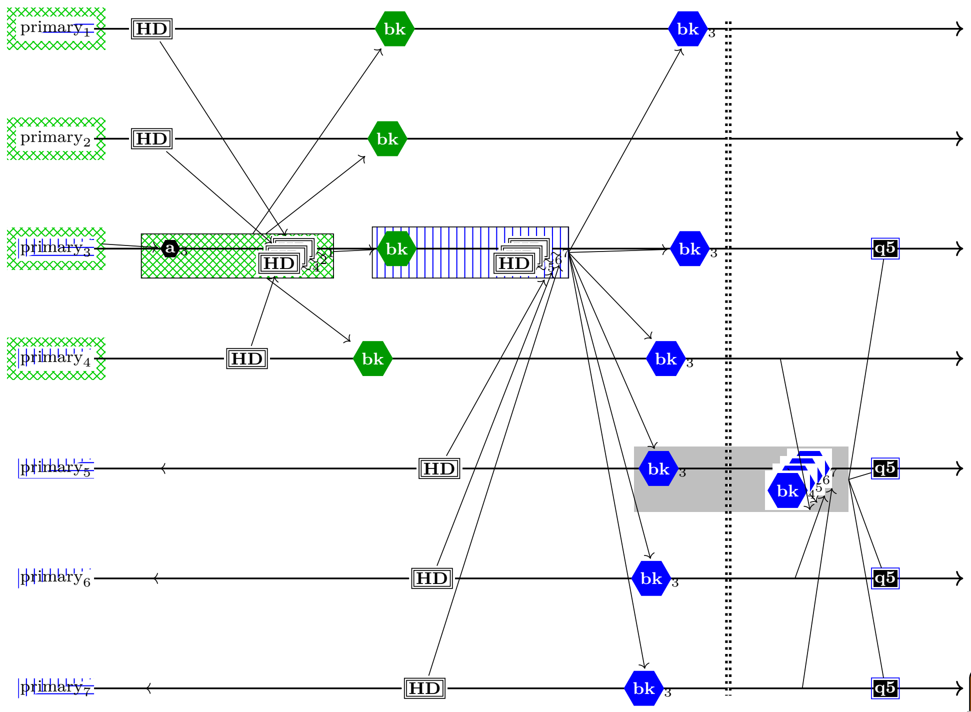
Block broadcast bk The aggregated signatures of a block header form a *learner-specific block*. The signature aggregator will broadcast to all primaries that belong to some quorum of the respective learner. (Later, these will be used as references to previous blocks in the learner-specific DAGs see **??**) 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 Fig. 6, for the difference between headers at genesis and the typical phase).

blue blocks go to all validators in (1) the signing quorum (2) all validators that are in some blue quorum

7.3 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



for cooler patternage
https://tex.stackexchange.c

Figure 5: The integrity protocol (concluding each round that's was "opened" in the availability-protocol)

Under certain conditions, in particular if there is exceptional delay for a specific learner, one can forgoe announcing a proper signed quorum and 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. In a similar way, the certificate of availability amounts to a checking of signatures.

describe in detail how
the checking of the availability of the headers takes place

7.4 Summary

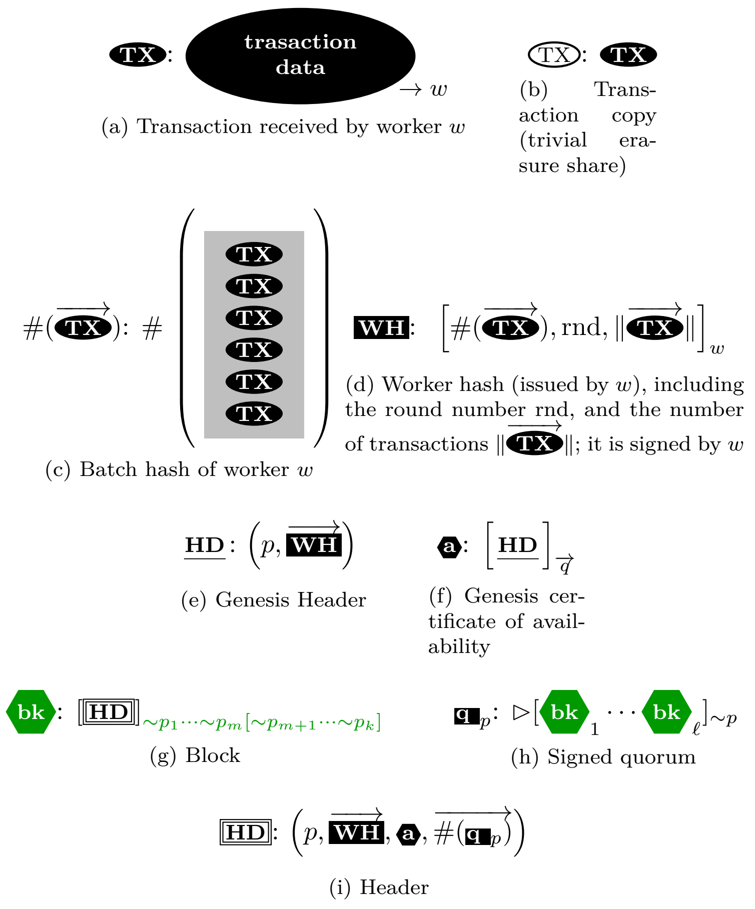
The availability protocol in a non-genesis round only differs in having

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 availabl).

discuss terminnolgy

8 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 6: 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.

[Her18] Maurice Herlihy. Atomic cross-chain swaps. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, PODC '18, page 245–254, New York, NY, USA, 2018. Association for Computing Machinery.

Notes

. Logging is terminology borrowed from César Sanchez: he likes to talk about *log-chain*s.

. though additional validators might want to join for the purpose of bridging for the purpose of chimera chains

. for "quorum-pairs" between two chains, explain why it is still much cheaper than having a merged super-chain (if it is ...)

. This matter should be discussed in the context of P2P.

. elaborate on the distribution scheme, and how the destination of each earasure share is determined (for the general erasure coding).

. Is there any such thing as "validator round"?
- sequence number
- validator height
- ...

. For general erasure coding: *Does this need to includes for each receiving worker, the (hashes of the) erasure coding shares that they should have available. ?*

. ... at least in theory, we'll keep it slick for the moment

. Somehow it seems overkill to have (hases of) signed quorums in the headers.

A food for thought

A.1 new

A.1.1 some of Tobias' brainstorming

how can we make chimera chains safer?

Requirement candidates

- *absolutism:* if a transfer between two base ledgers is commited in a "healthy" chimera chain, then the transfer is also (essentially) commited in both ledgers , i.e., eventually it will go through, unless we've got a "liveness fault"

whatever a liveness fault is

- *justified absolutism:* each consensus on an anchor block selection in a chimera chain needs "double consensus" from both chains, i.e., effectiverly, both chains need to approve the anchor block, which might lead to liveness issues ?
- *absolutism grounding:* does each anchor block in the chimera chain have also references to blocks in the underlying chain, "super-supporting" the anchor?
- *user authorization:* users may choose to effectively use the chimera chain as a limited bridge, "temporariliy" moving (some of their accounts) to the chimera chain

what's the difference to the bridge then here?

- *user's ruling:* moving assets into other chains – e.g., chimera but not necessarily – for atomic settlement on the other chain

A.2 old

A.2.1

- lots of state is completely independent of each other, can we use this for optimization using this "concurrency"
- how does this state right tool work ?
- How do the message graphs of HP compare to Narwhal DAGs

Isaac's thoughts

- message sharing (on the primary level) might be slow
-

the specs

A.3 very old, but ... very researchy

is reliable broadcast in narwhal actually even more useful? does Narwhal make consensus easier in that it restricts the set of valid values to propose? In particular, proposals of non-existing blocks are impossible? cf. ideas in Asynchronous Byzantine agreement protocols

what can be gained by randomness? at what price? well, just for the record, this is an interesting venue for (much) more performance, which is probably much harder to analyze ...

- Verification of randomized consensus algorithms under round-rigid adversaries
- *Randomized Byzantine Generals* Michael O. Rabin, 1983

B specific questions

- "availability certificate Availability Certificate: an aggregation of signatures from a Weak Quorum attesting that everything referenced by a particular Header is available. **Must include a signature from the Header's primary.**"
  - `Where is this signature?`
  - `What does it sign?`
- **my acknowledgment, "Availability Vote"** –

  ```
  isn't it rather a **promise/pledge**
  ```
- **about "blocks"**

  ```
  does it make sense to call these learner blocks, possibly even certified learner blocks
  ```
- **about TXs:**

  ```
   a learner might actually ignore a big chuck of transactions?
  ```
- **about the "two" protocols**

  ```
   if the integrity protocol gets stuck,
   the availability protocol will stuck?
  ```

  yes

  ```
     also vice versa ?
  ```

  partially
- **more about blocks**

  ```
  headers have "sequence numbers" (stemming from the creator)
  ```
- **about signed quorum**

```
would it make sense to call these
signed_certified learner blocks_

WHEN ARE BLOCKS broadcast and by whom?!
```

it *is* a broadcast and it is performed

- **after completion (in analogy to availability certificates)**
- **by the block creator**

—

C Random snippets

- Besides the learner graph, we assume that each transaction is relevant only to a subset of the learners.
-

—

New header construction and broadcast The following conditions will trigger the production of a new header. primary $\Rightarrow$ primary

- Each worker on the primary's validator has provided a worker has for the validators current round.
- If not genesis round, ...
- If not genesis round, ...

Once, the header is constructed (but NOT signed?), the primary broadcasts it to all other primaries.

Header checking and acknowledgment (Availability Vote) A received header of a primary is checked, according to the following check list.

- The worker hashes of the header must have been "uploaded" by the local workers (which are trusted to have check theses worker hashes).
- If not genesis round, ...
- If not genesis round, ...

After checking all this, the primary sents an acknowledment back to the header producer, *i.e.*, a message with the header signed.

Availability Certificate generation and broadcast After receiving a global weak quorum of header acknowledments for a previously broadcast header, the received signatures are aggreagated **and singed**. primary $\Rightarrow \forall$primary

The result is broadcast to all primaries.

Waiting for (certified learner) blocks $\boxed{\text{confusing switch to integrity protocol!}}$

Announcing signed quorums Once a validator has enough "fresh" blocks (see below) for a learner from a "preceeding" round, the primary signs a learner-specific quorum of such blocks. However, under certain conditions, it might be useful to anounce *empty* signed quorums, indicating that the next block header will not include any signed quorums for a specific learner.

C.1 Primary actions in the integrity protocol

Uniqueness attestation / Integrity Vote When a primary receives a header from another validator with a round number that directly succeeds the last known header of the sending validator, (for a new *"round"*) the primary signs that header (additionally for the purpose of integrity) and sends it back to the creator of the header. primary → primary

Certified learner blocks (Integrity certificate for headers) Upon receiving a learner-specific quorum of integriy votes for a header, the primary aggregates these and broadcasts a certied learner block. primary ⇒ primary

Signed block quorums

C.1.1 Genesis round

Header compilation Whenever a full set of worker hashes from another validator has , the primary primary → primary

—

learner-specific round numbers

- Each learner might "observe" different round numbers.
- In first approximation: do not expect any synchrony whatsoever!

reference a quorum of blocks from the previous round This is seen as follows, (for non-genesis blocks):

- the block contains a header $\boxed{\boxed{\textbf{HD}}}$
-

learner-specific DAG structure

D Erasure coding

The worker that has received a transaction from a client generates a suitable erasure code, broken up into a finite set of shares[7] to be distributed over all validators. The share distribution has to be such that any quorum of validators (relative to any learner)) can re-construct the transaction data from the set of shares that they obtained collectively. check this:Moreover, the map from erasure coding shares of the transaction to (mirror) workers that receive the respective share is determined by the worker's index (and the identifier of its validator).

cf. knowing whether we have all the shares ?

E State partition and fractal instances

Learners, *e.g.*, execution engines, want to be responsible for *changes* to the smallest possible part of the state. However, to enable basic actions such as cross-chain transfers, learners have to gather enough information about the global state to determine whether a given transaction TX is actually valid / executable. The matter becomes delicate if a transaction depends on parts of the state that different learners are responsible for. No learner can single-handedly determine if a given commited transaction is executable, unless some learner is tracking the complete global state—an almost impossible task already our days! Now, let us focus on transactions whose "inputs" (or "outputs") are spread out over several learners~~ as these are the trouble makers~~.

On chimera chains:
* "lock" the relevant parts of the state on all involved learners [a]
* "spawn" a chimera chain
* "move" locked state fragments to chimera chain
* "settle" on chimera chain
* "finalize" chimera chain
* "re-import projected state"

[a] footnotes in `\todo` notes have to be `\protect`ed, unless we put a caption.

We assume that each transition carries enough information to instantly deduce which part of the global state is accessed, possibly distinguishing between reading and writing.

F Old Intro

Motivation 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. Moreover, we want that any number of participants, holding assets on several *different* ledgers to "interact" "directly".

explain "interact" "directly"

[7] Shares are also known as *chunks*.

argumentation flaw: if we have a base ledger count of n, we will have a quadratic number of chimera chains as well, i.e., (at least) one learner for each chimera chain.

Probblem statement `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.[8] Can we operate a protocol that strikes a good compromise between the number of validators necessary to order (and execute) transactions, not spending more ressources than complete pairwise bridging?

the above does not cut it

G Some ideas for how to explain things

G.0.1 The risky fast track

The opposite extreme consists in not updating any (liveness) quorums and just hoping for "enough" overlap of assumed to be live quorums: however, it might be the case that the intersections do not hold enough honest validators. In the extreme, none.

G.0.2 First assessment in view of validator overlap

In view of a large overlap, note that both scenarios gain in practicability: if every pair of vaildator quorums has a large overlap, the increase in communication complexity is mitigated. Similarly, for the fast track, if overlaps are of validator quorums are big enough, we have a good chance to have at least one correct validator in their intersection.

G.0.3 Growing the overlap

A natural question is how one could go about getting a bigger overlap between two chains, for the sake of additional safety. In fact, one could imagine some sort of "auditing" nodes, which make it harder for Byzantine validators to trigger safety violations [**?**, opodis20HPaxos]

G.1 Tobias's corner

G.1.1 excerpting the blog post

The intuitive goal combo block

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

```
  [rb]
  /  \
[r]  [b]
 |    |
 |    |
[r]  [b]
 |    |
 |    |
[r]  [b]
 |    |
```

s.t.

both [chains, r&b,] agree to [unanimously] choose or [jointly] ignore the combo block.

Fortunately, when there is a lot of overlap between validator sets of different chains, these assumptions can become very reasonable.

within *same* consensus

undefined

bisimulate

clashes with bisimulation ?!

It’s Kind of Like a Bridge

In some ways, our Heterogeneous Consensus could create something like a trusted bridge: it allows operations across multiple chains, under specific trust assumptions. Other bridge designs tend to be based on multi-phase commits, so ours will be faster and will not require locking assets.

green quorum is sufficient to make green chain decide on a block without communicating with anyone else

now, a quorum suddenly seems independent of the green learner

Note that this would not be a problem if every red quorum had an intersection (featuring an honest participant) with every green quorum: red and green could avoid deciding contradictory things.

check: there’s no global weak quorum and heterogeneous narwhal does not help either ?

Fortunately, we have some promising directions for solving this conundrum.

e.g., ?

anyone following only blue transactions or only red transactions will not observe the fork

unobserved forks: sounds terrible!!

”move state” from, say, the purely blue chain to the blue-green chain when they want to do business with green smart contracts, and then back to the blue chain when they’re done.

how ? this sounds very much like a bridge, and does not solve the game theoretic problem?

Each piece of state (each smart contract) needs application-specific code to decide which chain holds its lock at which time.

how to keep track of this “locking table”

trusted by chains

the transactions are actually executed together

well, something like all execution engines do it (eventually?)

prove what messages are sent

proving what messages are sent *by whom?* users, validators, others, ...

G.1.2 On the blog post

Could we merge chains into chimera chains? in other words, why would we at all want to “split” again?

```
[r]  [b]
 |    |
 |    |
[r]  [b]
  \  /
  [rb]
  /  \
[r]  [b]
 |    |
 |    |
[r]  [b]
 |    |
 |    |
[r]  [b]
 |    |
```

random questions

not all learners equal, in yet a different way there are two very different type of learners:

- end-users: small stake, non-critical
- execution engines of validators: high stake, critical

on immediate finality which finality do we actually need ?

drawing on eigenlayer functionality? can we

what about dynamic graphs? all of

- validator sets
- assumptions
- learner sets

can change
cf.

Changing Quorums

who can initiate chimera chains / dynamic graphs?

rig up

tear down

on locks, not really, but ... what about garbage collection trigger for logged mempools

open issues Here is an open list of topics to clarify; in principle, these could be in the form of github issues.

(to be discussed)

certificates for honest behaviour examples, what's missing?

evidence for dishonest behaviour examples, what's missing?

garbage collection

- to do or not to do
- how to do / how to avoid

It's gonna be better than all bridges out there !! the connection to bridges has to be spelled out *explicitly*, in excrutiating detail

different perspectives the difficulty in describing—and understanding—TYPHON are the different perspectives that one has to keep in mind at each point in mind

- learner(s)
- validators / participants / workers
- designer
- attacker
- transaction batches ?!
- chains ?!
- something else ?

Thus, for every sentence, one might get it wrong, if it is not clear who's perspective we are taking.

what's shared batch looking like anywy?

moving state this is a phrase, which begs the questions

- who is moving the state?
- how is state partitioned?

Leaving the capability to move state with the user is also an extra task the user must be willing to perform.

"As Locks" `chains as holding exclusive locks on state`
this model transfers to "user's rule"

one of those statements to ponder about

> This would allow applications to "move data" to shared chains where atomic commits are possible, and then back. What we need is a specific interface indicating when such moves are safe, when atomicity in batches is preserved, and a way to easily move state from one chain to another.

best case scenario great
- that is a good example of quorums
- dosn't explain why atomic commit is possible

`Shared batch of transactions` now, what's a shared batch anyway? a batch of transactions in a combo block ?

- do chimeara chains need to acquire and release "ownership" ?
  -