Hetu: A causality network communication protocol - V0.04 AE

Abstract. Communication by leveraging centric service providers has been the key to support several categories of applications, including messagers, social networks, collaborative editing, etc. The trust of such systems relies heavily on trusting the service providers to obey the chronological order of messages, and it's infeasible to provide the trust with current systems. We propose Hetu, a P2P network communication protocol that provides verifiable causality among messages and events, to enable participants verifying that there's minimum anachronism behaviors in the system. The solution consists of a cohort network with efficient strong-locality, weak-consistency messaging and a hub which provides global consistency. The causality in cohort networks are tracked with a novel logical clock that achieves both space and time scalability. The system can generate computable proof for causality anomalies.

1. Introduction

Communications rely too much on centralized service providers to process messages. As trusted media, these service providers are overloaded with jurisdictions. This trust based model makes the communication system weak and fragile inside. Social interactions also suffer the rupture from this trust model of communication system. Since these centric entities cannot avoid mediating disputes, non-distorted messages are merely impossible. The cost of mediation actually increases communication cost, limiting the potential communication related services and generates much broader cost to make mini-distorted communications happen. With the possibility of distortion, the need for trust in the system diffuses. Service providers hassled their users by providing fake, hidden or changed messages. And there is no current mechanism to build up communications over a network channel without trusted parties.

To solve this problem, a distributed communication system based on cryptographic proof instead of trust is needed, allowing willing parties to communicate directly with each other without the need for a trusted third party. To prevent *anachronism* which can lead to information distortion and censorship, a key guarantee the system should provide is the correct and untempered *ordering* of events. In this work, we propose a verifiable logic clock protocol, Hetu, which provides strong causality guarantees in a large, open network. The protocol is significantly scalable for high-frequency events, and does not rely on trusted, centralized components. Hetu introduces a new logic clock construct, the Decaying Onion Bloom Clock (DOBC), that scales independently with the size of the system and can be used to accurately deduce the true causal relationship between events in the system. Hetu then leverages non-uniform incrementally verifiable computation to ensure untempered generation and distributed verification of DOBC clocks. Lastly, Hetu builds a hierarchical network architecture with smaller local cohort networks interconnected by a global hub network. Overall, Hetu provides correct and verifiable causal order of events in the network with unprecedented efficiency, enabling a large class of powerful applications with high scalability and minimum centralized trust.

2. Communication

A communication system delegates the interaction among participants into the individual interactions between each participant and the communication system. Participants must trust the communication system in such a way that they believe that even if they are interacting against the communication system, the effect of the interaction would be equivalent to them interacting with the other participants directly. However, in existing systems there's no universal mechanism existing to forbid the communication systems break the equivalence and to provide such trust. The communication systems take control of altering the eventual results of communication attempts, which may be against participants' wills. The way communication systems decide the result of communication is hidden from participants.

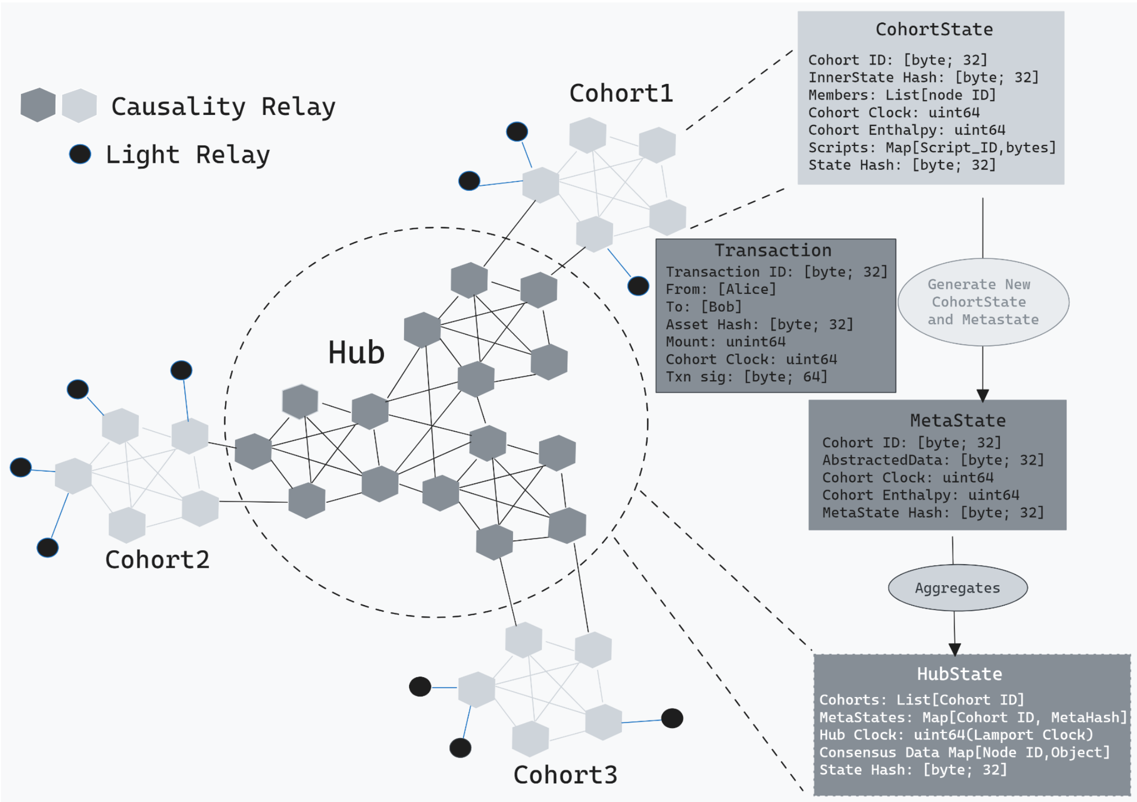
Causality as the universal representation of communication effects. Communication has different intentions, forms, protocols, etc. across various applications and scenarios. It could be hard to derive facts about communication with participant networks because e.g. the intention of communication does not specify the relevant participants. We introduce a graph of causality to solve this. Causality defines a partial order among events, namely "event A happened only because event B happened" or following the common terminology "event A happens after event B". In our system, the events become the communication events aka messages. One message can be the cause of multiple messages, and one message can be the result of multiple messages. Causing a loop is not impossible however. In other words, the messages and the causality among them forms a direct acyclic graph, of which only a single event without any predecessor.

When there's misbehaving in the communication system, ana-chronological causality can happen. Anachronism is the fault where the actual causality mismatches the causality that should happen. The locality of causality. Communication systems usually feature significant locality. A common pattern is a set of participants, which we called "cohort" in this paper, sending such kinds of messages that mostly only depend on the previous knowledge learned from the previous messages originated from the same cohort. As a result, the causality connections almost never escape cohorts, and the messages usually keep independent between different cohorts. This fact can be leveraged when designing communication systems, which leads to a hybrid communication system design: a cohort-level communication mechanism which exposes a fixed overhead on sending every message, and is good at tracking massive causality relationships efficiently and supporting dense-causality cohort communication, and a cross-cohort consistency, which is called "hub".

3. Network

To solve the anachronism problem of the network, a set of rules of events is designed in the cohort-based network,working together to ensure the network consistency.

- **Hub**: The Hub serves as the global network state cluster, aggregating meta states of cohorts to form a global landscape encapsulated in the hub-state. The hub leverages Lamport Clock to maintain a global order across the entire network.
- **Cohort**: A cohort is the autonomous, localized network cluster, maintaining its own states encapsulated in the cohort-state. The innovative vector clock is used for managing the partial order of states, allowing for a detailed view of the history of network events.



There are three types of states: Cohort State, Hub State, and Meta State.

1. **Cohort State**: Cohort state records key events within a cohort.It updates continuously with new transactions and operations. Cohort members can set specific rules and functions that are encoded into the cohort state.New nodes can either create a new cohort or join an existing one by communicating with the cohort and synchronizing the latest cohort state.
2. **Hub State**:The Hub State, managed by the causality-hub, provides a global causality view of the network.It ensures network consistency across the global network and allows different cohorts to interact and synchronize.The Hub State is formed by aggregating meta states from all active cohorts.
3. **Meta State**:Meta State acts as a bridge between the cohort state and hub state, ensuring synchronization and consistency.It provides the Causality-Hub with condensed information about a specific cohort.Whenever a transaction or state update occurs within a cohort, it's aggregated to form an updated meta state.These updated meta states are then aggregated into the hub state, ensuring the hub state reflects a complete and up-to-date view of the entire network.

In the Hetu protocol, we can define the following state transition functions:

1. TRANS_COHORT(S, TX) -> S': This function describes the state transition at the Cohort level.
    1. Check if the transaction format is correct and if the signature is valid. If not, return an error.
    2. Calculate network fees and deduct them from the sender's Cohort State. If there are insufficient resources in the Cohort State to cover the fees, return an error.
    3. Execute transaction instructions on the Cohort State. This may involve modifying internal state information within a Cohort or performing specific operations.
    4. If the transaction is successful, update the Cohort State and encode transaction information into a new Cohort State.
    5. Synchronize this new Cohort State with nodes within a Cohort.
    6. After executing transactions, generate a new Meta State that includes summary information of transactions as well as logical clocks and consistent hashing for each cohort.
2. TRANS_META(S, M) -> S': This function describes how to update Hub State using Meta States.
    1. Check if Meta State format is correct. If not, return an error.
    2. Aggregate all information from Meta States into current Hub State. This may include updating global transaction history or updating specific cohort's state information.
    3. If transaction information contained in Meta States is valid, update Hub State and encode this Meta State's information into a new Hub State
    4. Update global logical clocks and consistent hashing in Hub States
    5. Synchronize this new HubState with cohorts in order for every cohort to have access to up-to-date global status information.

4. **Logic Clocks**

4.1 Causality Definition

Hetu concerns the causal relationship among objects in the network. We define generic *create* and *mutate* functions for all objects:

- *create()* $\rightarrow$ *o:* the *create* function generates an object in its initial state
- *mutate*(o_1, o_2, …) $\rightarrow$ *o':* the *mutate* function takes a list of objects and generates a new object o'

The exact semantics of two functions are defined by the applications. We define a binary relation $\lessdot$ on the set of objects in an execution of the network. $\lessdot$ denotes the causal relationship between any two objects, i.e., $o_1 \lessdot o_2$ if and only if o_2 is causally dependent on o_1. Not all objects have causal relationships. If neither $o_1 \lessdot o_2$ nor $o_2 \lessdot o_1$, o_1 and o_2 are concurrent. We use $/\!/$ to present concurrent objects (e.g., $o_1 /\!/ o_2$). Object causality in Hetu is defined as follows:

- If an object o is generated from *create*, o is not causally dependent on any other object in the system, i.e., $\forall$ o' $\in$ O, o' $\not\lessdot$ o, where O is all objects ever generated in the execution.
- If an object o is generated from *mutate*(o_1, o_2, …), o is causally dependent on o_1, o_2, …, i.e., $o_1 \lessdot o$, $o_2 \lessdot o$, …

The $\lessdot$ relation in Hetu has a stronger definition than prior work [1, 2, 3]. Instead of “possible influence”, $\lessdot$ implies definite causal relationship between two objects. More formally, if $o_i \lessdot o_j$, there exists a sequence of *mutate* invocations such that $mutate(o_i, \ldots) \rightarrow o_1$, $mutate(o_1, \ldots) \rightarrow o_2$, …, $mutate(o_n, \ldots) \rightarrow o_j$.

4.2 Hetu Logical Clock Construct

We use logical clock [1] to deduce causal relationships ($\lessdot$ defined in the previous section) in the Hetu network. We similarly define a binary relation, <, on the set of logical clocks. < is also a partial order, i.e., not all logical clocks are comparable under <. Each object o_i in Hetu is tagged with a logical clock, represented as C_i. An object o_i with state s_i and logical clock C_i is therefore represented as a tuple (s_i, C_i). The key property of our logical clock is that it should *characterize* causality, i.e., for $o_1 = (s_1, C_1)$, $o_2 = (s_2, C_2)$, $C_1 < C_2 \leftrightarrow o_1 \lessdot o_2$.

The Hetu logical clock is based on the Bloom clock (BC) [4]. The BC uses the counting Bloom filter [5] to probabilistically determine the causality between objects, and is represented by a vector of n integers $[c_1, \ldots, c_n]$. In the context of Hetu, when an object $o_{i+1} = (s_{i+1}, C_{i+1})$ is generated from another object $o_i = (s_i, C_i)$, i.e., $\mu(s_i, C_i) \rightarrow (s_{i+1}, C_{i+1})$, we calculate the new clock C_{i+1} in the following way: We use a family of m cryptographically secure hash functions $h_1, \ldots, h_m$ that produces m indices $h_1(s_{i+1}), \ldots, h_m(s_{i+1})$. Each index is then mapped and incremented on C_i to produce C_{i+1}. μ can either be the *create()* or the *mutate()* function. Since *create()* does not take any existing object as input, we use the zero clock value $[0, \ldots, 0]$ to derive the clock for the output object.

The rule when comparing Bloom clocks is as follows:

- $C_x < C_y \leftrightarrow \forall c_{xi} \in C_x, c_{yi} \in C_y, \exists c_{xj} \in C_x, c_{yj} \in C_y : c_{xi} \geq c_{yi} \land c_{xj} > c_{yj}$

Eventually, a BC will increment to a point in which comparisons between two clocks will always lead to a false positive due to hash collision. To address this issue, Hetu introduces a new logical clock construct, the Decaying Onion Bloom Clock (DOBC), with the following properties:

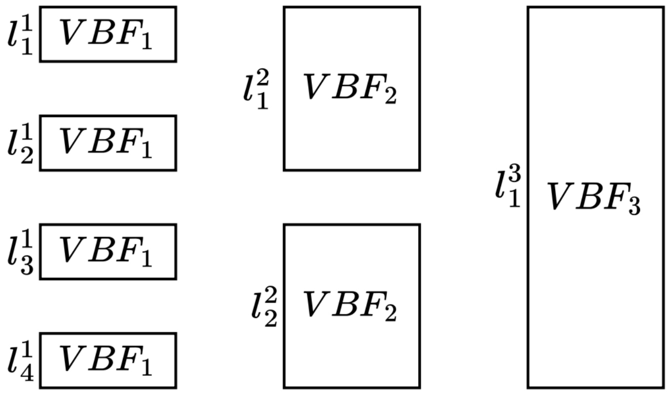
- DOBC probabilistically determines causality between objects with a depth difference of at most k.
- DOBC keeps a finer grain memory of recent state transitions. This is opposed to distant state transitions, where its view is compressed to produce a coarser grained expression. To provide indefinite utility across any number of state transitions, DOBC eventually forgets states that are too distant.
- A sub-function that allows DOBCs of different depths to be merged. The causality utility is maintained with regard to any of its ancestors.

We generalize the Counting bloom filter construct to variable-sized Bloom filters (VBF_i), where each of its n indices are stored with exactly i bits. The DOBC also consists of $|L|$ layers, each layer l^i stores a predetermined amount $|l^i|$ of VBF_{ji} s, where j^i is the size an index for each VBF at layer l^i and $j^i > j^{i+1}$. For the sake of simplicity, let's assume $j^i \cong i$. Each $VBFi$ in a layer is ordered from $l^i_1, \ldots, l^i_{|li|}$.

Suppose for a certain execution path, it produces an ordered set of objects $\{o_0, o_1, o_2, \ldots\}$, where $\exists \mu$ *in M*: $\mu(o_i = (s_i, C_i)) \rightarrow (o_{i+1} = (s_{i+1}, C_{i+1}))$. Initially, VBF_is on all layers are set to 0. For illustration purposes, let's use the following settings:

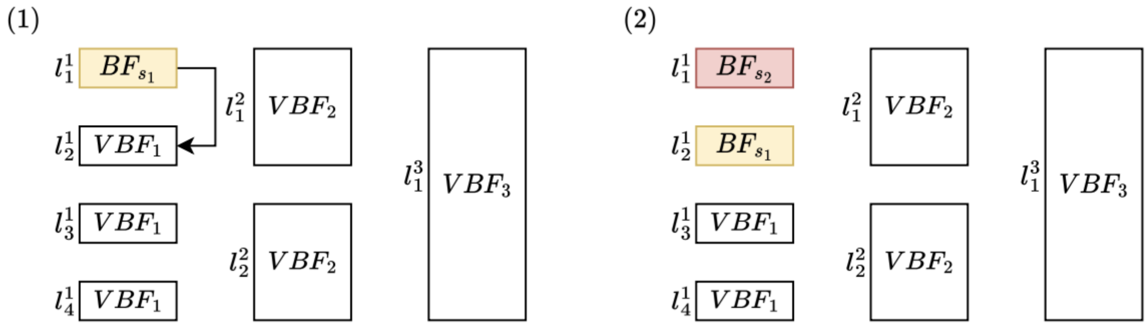
$$|L| = 3, |l^1| = 4, |l^2| = 2, |l^3| = 1$$

This is illustrated in this figure:

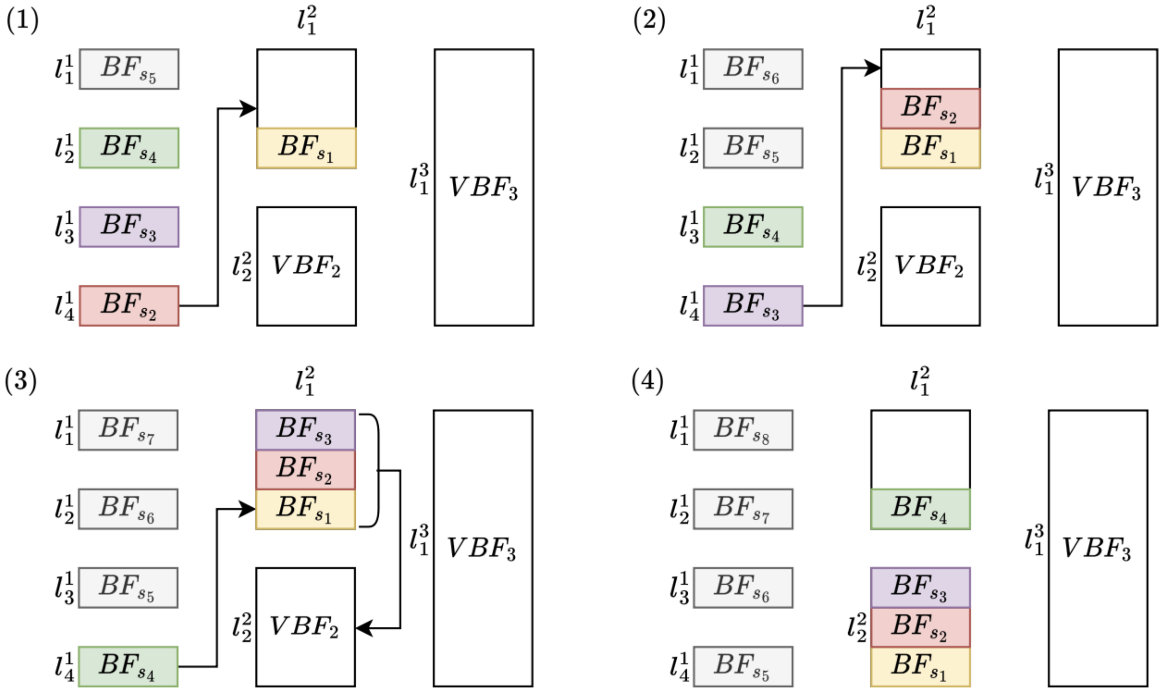


When o_1 is generated, a Bloom filter BF_{s1} is created by hashing s_1 with the family of m distinct hash functions and setting the corresponding indices to 1. It is important to note that $VBF1 \cong BF$ if both contain the same number of indices.

BF_{s1} is inserted into $l^1{}_1$. When s_2 is reached, BF_{s2} is created and placed into $l^1{}_1$, and BF_{s1} is moved to the next available slot (which in this case is $l^1{}_2$). DOBC for o_1 and o_2 is illustrated below:



Eventually, as new objects (and states) are created, in the DOBC for a specific object (o_4), BF_{s1} will be at $l^1{}_{|li|} = l^1{}_4$. To make space for BF_{s4} , BF_{s1} instead moves to $l^2{}_1$. In theory a VBF_2 can hold the compressed information of $2 * 2 - 1 = 3$ VBF_1s.Therefore,BF_{s1}, BF_{s2}, BF_{s3} are added together before it moves to $l^2{}_2$. This is illustrated by the figure below. Intuitively, a $V\,BF_{i+1}$ in layer l^{i+1} can store a multiple of VBF from the previous layer (l^i).



When $l^{|L|}_{|li|}$ has reached the maximum capacity, and a new state is reached. For the new object, $l^{|L|}_{|li|}$ is deleted and $l^{|L|}_{|li-1|}$ or $l^{|L|-1}_{|li|}$ takes its place. In the context of our example, BF_{s1} , …, BF_{s6} is evicted from $l^3{}_1$ and BF_{s7}, ..., BF_{s9} takes its space.

In DOBC, we only keep a limited history k of states, therefore only histories of a certain range can be compared. The greater the overlap, the lower the possibilities of false positives. We will utilize the same setting from above to illustrate an example. Suppose we have the DOBC for o_{18} and o_{16}, we determine the causality of o_{18} on o_{16} as following: Since each state has differing depths, we compare different sections of its DOBCs to draw our causality conclusion. For example, $l^1{}_3 \in o_{18}$ should correspond to $l^1{}_1 \in o_{16}$. Similarly, $l^2{}_1 \in o_{18}$ corresponds to addition of $l^1{}_3 \cap l^1{}_4 \in o_{16}$. Intuitively, two sets of VBFs are comparable between two DOBCs if they correspond to the same depth. If all comparable VBFs in o_{18} are greater than or equal to the corresponding VBFs in o_{16}, then we draw the conclusion that $o_{18} > o_{16}$ with some acceptable probability.

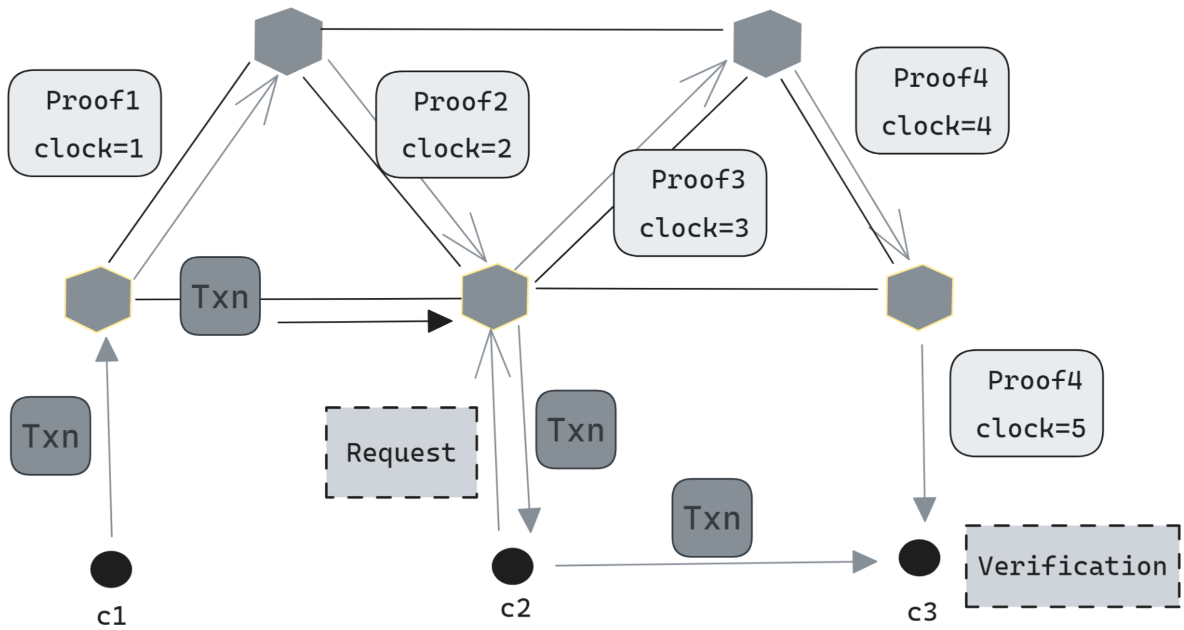
5. Causality Relays

The vital role of the Causality Relays is to maintain the network's integrity and facilitate mini-distorted communications. Their essential functions include:

- **Relay-based Communication**: Emphasizing the importance of Relays, they act as the protocol's backbone, enabling the free and efficient interaction with nodes. These Relays are responsible for the creation, storage, and management of data, establishing a resilient and secure communication system. There are two types of relays, namely causality relay and light relay. Causality relay is mainly responsible for generating causal proof, while light relay can support causality and propagate network data and status.
- **Heart-Beat**: Unlike traditional heart-beat, each causality relay will regularly send Heart-Beat with logic clock to its neighbors, and then this neighbor will add the logic clock to the

Heart-Beat and pass it to its next neighbor, thus establishing asynchronous and interlaced causal connections within the Cohort.

- **Scripts**: The Causality Relay allows developers to deploy self-verifying Scripts within the network. These scripts, each with their unique, facilitate the efficient transmission of information, ensuring a reliable, efficient, and secure environment for information exchange.
- **Hetu Identity**: The generation of Hetu identity results from the establishment of causality. When a logical Cohort joins the protocol, it creates a causal relationship connected to the Hub and generates a global Global-ID and a local Cohort-ID during the creation of the causal relationship.



6. Messaging

Features as a coordination system. Every causality relay has a N-dimension coordination (we use 2-dimension (x, y) for illustration here). Each dimension identifies the messaging preference of the causality relay in some way. For example, all relays that are interested in topic A will have a coordination close to (x_A, y_A). If some relays have equal interest in two topics, they may choose a coordination that is close to the middle point of the two center points. Unlike ID, relay's coordination is volatile. To update the coordination, a relay notifies the relays near the new coordination through the underlying Kademlia network.

Coordination-based, distance-aware gossip. When sending a message, the causality relay decides how much "energy" is consumed to send the message. The more energy is used, the wider the message will be spread. Sender may also configure the "direction distribution" of the sending energy through a series of weight vectors in the coordination system, but in most cases relays will want to send to every direction homogeneously.

The energy of the message reaching a causality relay decays according to the distance the message gets traveled. In this transport network the energy is “quantum-ized”, so only finite relays can be reached with non-negligible energy. The receiving relays then make a decision: if they are interested in the message, they consume the reaching energy and the message; otherwise, they rebroadcast the message with the reaching energy. The key is that the sender decides (and gains) the message energy, while the receivers pay the energy consumption. The receiving relay accepts the message it wants to build causality based on the message. A “blackhole” relay which only consumes energy but does not contribute to the causality will be penalized by the rule. If the receiving relay builds causality, it has to do so by later sending out messages, which will consume the energy it receives.

Messaging, causality and incentive. Messages that contribute to causality are meaningful and the system health is beneficial from them, so such messages will be allowed to be sent with more energy. Further incentive will encourage relays to build more causality, which will in turn reveal what messages they are consuming and should pay energy for. Effectively, the causality graph partially serves as a ledger of energy (for cohort-local messages).

Messaging and cohort/hub. As mentioned before, the messaging network is decoupled from the cohort/hub architecture, but most of the relays of a cohort will normally reside closely to reduce messaging energy. Only a few “gateway” relays who communicate between cohort and hub may be the exception. Hub messaging happens in a predefined sub-space in the coordination system. They never escape the sub-space, and dedicated rules are applied when transmitting them. Moreover, developers who have control over the main relays of a cohort can simulate any form of unicast/multicast/broadcast of cohort-local messages by fine-tuning the coordination and the interest of the relays, although the network only supports gossip.

7. **Security of Hetu**

7.1 Verifiable Logical Clock

Adversaries in the network can break the causality guarantees provided by Hetu by deviating from the Hetu logical clock. Three types of attacks can be launched by an attacker:

(1) Forging of Causality. Attackers can perform two types of forging attack. For the first type, even if an object o_j has no causal dependency on o_i, an attacker can forge such causality by modifying the logical clock value of o_j such that $o_i < o_j$. Secondly, suppose $o_i \prec o_j$, an attacker can forge a wrong causality relation $o_j \prec o_i$ by generating the logical clock value of $o_j < o_i$.

(2) Loss of Influence. Attackers can hide causality between objects. For instance, suppose $o_i \prec o_j$. An attacker can modify the logical clock value of o_j such that $o_i \parallel o_j$.

To address the above attack vectors, Hetu uses verifiable computation. Verifiable computation is a cryptographic construct to verify the result of computation when identities who are performing the computation can not be trusted. Succinct Non-interactive Arguments of Knowledge (SNARKS) are created to address this issue. The goals of SNARKS are quite simple. For a given statement: $\mu(x) \to y$, it produces an accompanying proof π. This proof can be verified to assert that the statement is true with all but a negligible probability.

SNARKS are useful in distributed computing, as it allows a verifier to validate computationally expensive function executions in a tiny fraction of the time to run it. However, SNARKS as it is, is insufficient to be utilized with Hetu. Consider that we want to verify a chain of executions, where a state n, provides a valid proof that for the transition from $\mu(n-1) \rightarrow n$. Although, each step can be verified easily, the size of the proofs grows linearly to n. In a highly evolving and volatile system, this growth in proof size is unacceptable.

A recursive proof system addresses this issue by having the proof be of a constant size regardless of the depth in the chain of executions. More concretely given a set of mutate functions $M = \{\mu 1, \cdots\}$, $m_n = \{\mu_{n1}, \mu_{n2}, \ldots \mu_{nend}\} : m_n \subseteq M$, $\mu_{nend}(\ldots \mu_{n2}(\mu_{n1}(s_0))) \rightarrow s_n$, where s_0 is the genesis state, and $\forall i, j \in n$, $|\pi_i| = |\pi_j|$. The time-cost to verify should also approximately be the same.

There exists many proof systems or SNARKs, each with its unique characteristics. However, in the realm of recursive SNARKs, there exists mainly three categories:

(1) IVC: Incrementally Verifiable Computation (IVC) by Valiant is the "father" of recursive proof systems. Its creation led to the various recursive proof systems we have today.
In the context of Hetu, given a genesis state s_0, the next state can be created by applying a function $\mu \in M$, $|M| = 1$. That is to say the n^{th} state is the result of applying the function on s_0 n-times. At each state, a proof of constant size can be generated that asserts its validity. IVC is not applicable in Hetu as it dictates a particular static chain of executions.

(2) Non-Uniform IVC: First introduced in SuperNova, a non-uniform IVC is described as a generalization of IVC's in respect to the family of functions M. That is to say now $M = \{\mu 1, \ldots,\}$, $|M| \geq 1$.
Additionally, it introduces a new function φ that takes in the inputs of the function $\mu_i(s_{i-1}) \rightarrow s_i$ and a potentially non-deterministic input to output next function $\mu_{i+1}(s_i) \rightarrow s_{i+1}$.
Intuitively, based on the existing state and the function φ, φ creates a program by selecting a particular ordered multiset of functions from M.

(3) PCD:
Proof carrying data (PCD), is described as a generalization of IVC's single chain structure to a directed acyclic graph.
In the context of our system, PCD will allow the creation of proofs for a given state s_i. The proof asserts that there exist a particular set of functions m such that:
$M = \{\mu_1, \cdots\}$, $m_i = \{\mu_{i1}, \mu_{i2}, \cdots \mu_{iend}\} : m_x \subseteq M$, $\mu_{iend}(\cdots \mu_{i2}(\mu_{i1}(s_0))) \rightarrow s_i$.
Note that the set of functions m is not necessarily hidden. A valid PCD proof just asserts that such a set exists.

In Hetu, each object only contains a state, a corresponding clock and depth. This means, any malicious user might create fake clocks to imply false casualties. To protect against such tampering, a verifiable proof π_x is included to attest to validity of s_x, C_x. That is to say that the mutate function has been applied correctly at each step: μ_{xend} $(\ldots \mu_{x2}$ $(\mu_{x1}$ $(o_0))) \rightarrow o_x$ starting from the genesis object.

Simply asserting that a particular mutation step is insufficient $\mu_x(o_{x-1}) \rightarrow o_x$. This is because the mutations leading to o_{x-1} from the genesis object must also be verified. Therefore, Hetu requires a recursive proof system that does not simply verify the validity of an object o_x, but it has been mutated from a valid object o_{x1} as well. There are two ways to apply recursive proof systems to Hetu. First, we can use Non-uniform IVC. To determine if non-uniform IVC's are usable with Hetu, we need to determine if φ is non-deterministic at each depth. That is to say if for a given object o_i, it is possible $\varphi(o_i$, some input$) \rightarrow \{\mu_1, \ldots\}$. Second, PCDs are exactly what we require in Hetu. However, the theoretical performance is substantially worse compared to existing proof systems or recursive proof systems like SuperNova. Therefore, we would have to find a more efficient variant.

7.2 Cohort-Level Causality Dissemination

Verifiable computation can effectively address the causality forging and loss of influence attacks in §7.1. Any participant in the system can independently verify the true causality, or true causal independence between any two objects. Formally, by comparing the verified logical locks, they can correctly infer the correct causality (or lack of), i.e., $o_i \prec o_j \leftrightarrow C_i < C_j$. However, VC alone is not enough for defending another type of loss of influence attack. Under this attack, a participant receiving an object can pretend it has no knowledge of the object. This is commonly known as the *omission failure.* Omission failure can also lead to a weaker form of loss of causal influence, since subsequent actions by the recipient can be influenced by the object, but it pretends to have no causal influence. Its future actions also do not violate the computation rules, so VC is incapable of detecting such behavior.

To address this issue, we require nodes in a cohort to disseminate causality information to ensure omission failures can be detected. For instance, when node n_1 sends an object o_i to node n_2, to guarantee that n_2 does not omit the knowledge of o_i, n_1 will require n_2 to acknowledge with an object o_j such that $o_i \prec o_j$. n_1 then can use $o_i \prec o_j$ as a *proof of causality*, and disseminate the proof to a majority of nodes in the cohort. By generating objects o_k causally dependent on o_j ($o_j \prec o_k$) and interacting with n_2 using such objects, node n_2 can no longer hide the influence of o_i.

8. Causality Applications

With Hetu, different types of causality protocols or applications will be empowered. First ones are communication-related, messengers and social network protocols, which need to deconstruct the trust model of these scenarios and generate more elasticity over censorship or message manipulations. Also, the second ones are transaction-related protocols, which could leverage Hetu to build up new weak consensus to minimize the trust through high frequency local-related transactions. The third ones are governance-related applications or protocols, like oracles, sequencers, DAOs, which could provide a substitute for the global order of the system.

I. Causality Messengers

The messengers based on the traditional communication system cannot guarantee an anti-censorship messaging application actually. Too many trusted parties are mediating the messages delivering process. Leveraging Hetu protocol we could design an anti-censorship messenger which is running on a relay network. The key of causality messengers is to provide the causality relationship of messages transparently with the users. The purely privacy could also be guaranteed in these messengers by the aid of cryptography methods.So the causality messengers could provide more social networking services based on these local casualties.

```
// Pseudocode for Anti-censorship Messenger based on the Hetu Protocol
// Create a new message
 let (encrypted_message, logic_clock) = encrypt_with_logic_clock(message);
// Send a message
relay_network.send(logic_clock, encrypted_message);
```

```
// Receive a message
let (logic_clock, encrypted_message) = relay_network.receive();
if is_valid_logic_clock(logic_clock) { // Validate the logic clock
  let message = decrypt(encrypted_message);
```

II. Causality Games

A new kind of Poker Game could be developed. The problem of poker games is that the records of players could be easily modified by centralized service providers and also the gaming record could not generate the general trust of users. Casualty games could happen which will allow people to play and compete in some local network, with strong consistency. And no worry about the centralized manipulation of the gaming process. A more transparent poker game could be possible.

III. Causality Social Network Protocols & Social Points System

A small world based social network could happen on Hetu. Surely, the content curation process will be more transparent and trust-worthy. Some strong relationships of social interactions could be redesigned on relay networks. With the local consistency of the logic clock, a social points system could be designed. This weak consensus could help some social networks to generate an initial social points system and internal incentive models.

IV. Causality Sequencer & EVM-Compatible Services

A more trustless EVM service for the blockchain industry could happen. Some of the services of EVM now need centralized entities to participate to ensure the efficiency, for example the sequencer. The causality sequencer could solve the key problem of these off-chain consistency problems.

V. Causality payment service for AI agents

Leveraging the power of a causality relay network, a more secure and reliable payment service can be established. The causality nature of the system could enable some AI agents to achieve a fast, secure, and cost-effective payment network,and no single entity has control over the entire network, promoting fairness and transparency.

In the progressive way , Hetu protocol may also empower some progressive applications with more imaginations, including Satellite Communication Protocols which can enable the more secure interstate communications. Also trading exchanges may not rely on the matching engines. Some serendipity engineering and new computable frameworks may generate the new organizations which could be a substitute for DAOs.

9. Conclusion

In this paper we propose a communication system, Hetu, that minimizes the trust placed by the participants to ensure non-distorted information. Hetu leverages causality to guarantee the correct ordering of events, eliminating potential anachronic anomalies . To do so, Hetu combines a new form of logic clock, the Decaying Onion Bloom Clock, and verifiable computation using non-uniform IVC in the cohort-level communication system to implement verifiable causality without central trust. A larger hub-level communication system then applies Lamport Clock. The strong guarantees of Hetu, as well as its high scalability, enables developers to deploy a wide range of large-scale applications, including messengers, social networks, games, and financial applications.

Reference

[1] Leslie Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. Communications of the ACM 1978.

[2] Fidge, Colin J. ``Timestamps in message-passing systems that preserve the partial ordering". In *Proceedings of the 11th Australian Computer Science Conference (ACSC'88)*. Vol. 10. pp. 56–66.

[3] Mattern, Friedemann. "Virtual Time and Global States of Distributed systems". In *Proc. Workshop on Parallel and Distributed Algorithms*. Chateau de Bonas, France: Elsevier. pp. 215–226.

[4] L. Ramabaja. The Bloom Clock. arXiv 1905.13064, 2019.

[5] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. Communications of the ACM, 13(7):422–426, Jul 1970.

[6] Miguel Castro and Barbara Liskov. Practical Byzantine fault tolerance. In Proceedings of Symposium on Operating Systems Design and Implementation, volume 99, pages 173–186, 1999.