Peer-to-peer Non-Consensus Partial Event Ordering Network

by Levente Mészáros at G

Overview

Goals

Build an information system where all participants can mathematically prove lower and upper bounds for the time of digital events generated by any participant such that the time interval is shorter than a predefined value (e.g. a minute) according to the *local clock of the participant*.

Requirements

- permissionless: allow anyone to participate without requiring authorization
- trustless: require as little trust in other participants as possible
- dynamic: allow rapid changes in participant relationships and network topology
- decentralized and redundant: avoid having a single point of failure
- distributed: allow efficient use of resources over the whole network
- consistent: prevent different participants having conflicting event ordering results
- disagreement supporting: allow anyone to form their own partial event ordering
- efficient in terms of memory, disk, cpu, and network bandwidth usage

Non-Requirements

- global consensus is unnecessary and would require sacrificing too much
- implicit transfer of value for publishing is unnecessary due to low resource usage

Components

- nodes are computing devices connected in a network
- events are data structures generated and stored by nodes
- clock events are special events which have local timestamps

Use Cases

- nodes can be run by people at home, companies at the office, communities, cities, courts, countries, governments, non-profit organizations, etc.
- nodes can be computers, routers, mobile phones, sensors, cars, IOT devices, etc.

- events can be tweets, articles published, photos taken, videos recorded, temperature measurements, noise level measurements, etc.
- nodes can validate the timestamp of events according to *their local clock*
- nodes can use the real world relationship between events (oracles) to even further validate the timestamp of events (e.g. an exploded bomb must be consistent with nearby noise level measurements according to *the local clock* of the validating node)
- business to business, business to government, patent and many other legal applications are possible where proving a lower and upper time boundary for an event to a 3rd party has important consequences (e.g. legal contracts, patent disputes)
- publishing digital events in the system is practically free, so nodes could publish as many versions of digital content as they desire

Notation

- nodes are denoted as N (or N_i)
- events are denoted as E (or E_i)
- timestamps are denoted as T (or T_i)
- clock events are denoted as C (or C_i)
- timestamps of clock events are denoted as T(C) or T(C_i)
- if E_i contains the hash of E_k then it is denoted as $E_k \leftarrow E_i$
- event chains are denoted as $\underline{E}$ (or as $\underline{C}$), and $E_i \leftarrow E_{i+1}$ holds for all E_i in $\underline{E}$ (or $\underline{C}$)

Operation

Running Nodes

- anyone can run as many nodes as they see fit
- nodes have private/public keys to prove identity
- nodes have neighbors, topology may change over time
- nodes share hash of clock events with current neighbors continuously
- nodes can share events (optional)

Creating Events

- any node can create any number of events any time
- events contain some content: picture, audio, video, text, etc.
- events contain some hashes (and locations) of selected local clock events
- events optionally contain a randomly generated salt
- nodes optionally sign events

Creating Clock Events

- any node can create any number of clock events any time (usually periodically)
- clock events contain local timestamp (*no global clock is needed*)
- clock events contain some hashes (and locations) of selected local events
- clock events contain some hashes (and locations) of selected clock events (both local and neighbor) that have timestamps smaller (but not necessarily for neighbors) than the timestamp of the local clock event
- clock events optionally contain a randomly generated salt
- nodes optionally sign clock events
- nodes broadcast clock event hashes to current neighbors

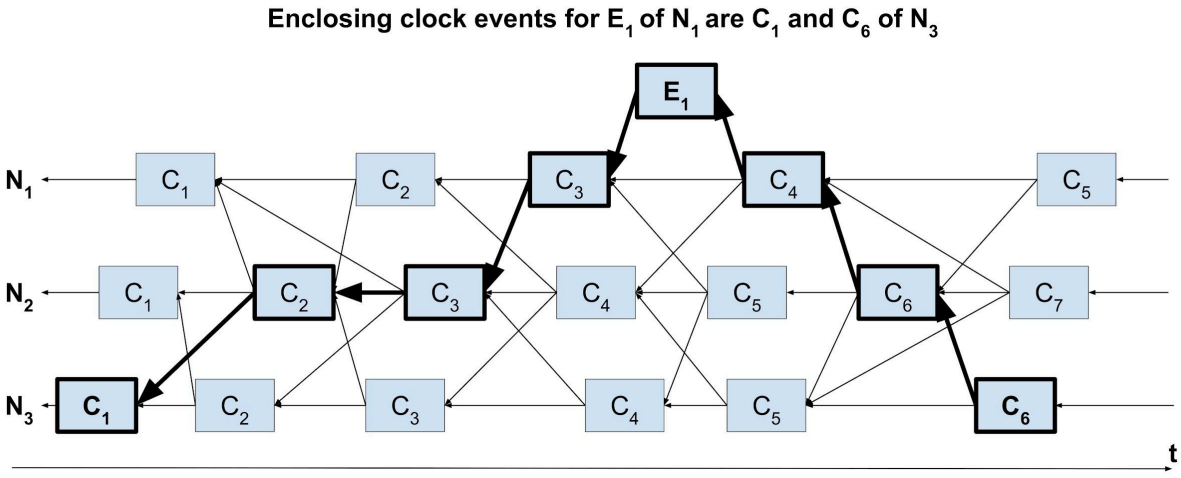
Proving the Order of Two Events

- to prove E_a is created earlier than E_b
- node shows one or more $\underline{E}$ such that E_a is the first and E_b is the last element of $\underline{E}$
- for example: $E_a \leftarrow E_1 \leftarrow E_2 \leftarrow \ldots \leftarrow E_{n-1} \leftarrow E_n \leftarrow E_b$

Proving Event Time Boundaries

- to prove any E is created between T_a and T_b according to the local clock of N
- node shows a C_a of N having $T(C_a) >= T_a$ such that E is created later than C_a
- node shows a C_b of N having $T(C_b) <= T_b$ such that E is created earlier than C_b

Partial Event Ordering Example



- E_1 of N_1 is created between $T(C_1)$ and $T(C_6)$ according to N_3
- C_1 is the lower bound of E_1, C_6 is the upper bound of E_1 according to N_3

- $\underline{E}$ from C_1 to C_6 can be found using hierarchical beam search with the help of N_2

Finding Event Time Boundaries

- nodes maintain a hierarchical routing table over time (similarly to IP routing)
- finding the lower/upper bounds is done with a probabilistic beam search recursively along the neighbor hierarchy
- during the search a neighbor is either the originator of the event, so it can easily provide the local lower/upper bounds
- or a neighbor is not the originator of the event, so it extends the lower/upper bounds that is received from the next neighbor towards the destination, using its own chain
- finding the boundaries may require transfer of value to provide incentives for node operators to respond to queries
- result is lower/upper bound $\underline{C}$ which form a single $\underline{E}$ with E included

Finding the Order of Two Events

- node finds event time boundaries for both events using the above method
- node compares the event time boundaries in order to determine the order of two events
- either one interval is strictly smaller than the other, or the order of the two events cannot be determined according to the local clock of the node

Estimating Network Efficiency

Network Assumptions

- few billion nodes (similar to the number of IP nodes in the internet)
- hierarchical (and redundant) ever growing clock event distributed DAG among nodes
- average distance of leaf nodes from top level nodes is ~10 hops
- average neighbors per node is ~10 or more (i.e. ~10 other clock events included)
- nodes generate events as content becomes available
- clock event hashes are distributed to neighbors
- events are not distributed by default
- nodes generate ~1 clock event per second
- clock events take ~100-1000 Byte disk space

Rough Approximation

- nodes store their own local events only
- nodes store ~3-30 GBytes clock events per year

- chain length between two C_a and C_b of N_i including an arbitrary E of any N is ~40 events long (2 * 2 * network depth due to hierarchical routing, there and back again)
- chain length from C_a to C_b excluding the content of E is roughly 40 * 10 * 100 = 40,000 bytes (fits into a single UDP datagram)
- timestamp of any E can be narrowed down to less than ~40 seconds by all N according to the local clock of N

Preventing Attacks

Postcomputing and Forging the Past

- to insert an event back in time as if it were there all along
- node creates auxiliary E_i containing a non-existent event hash at T_a
- node waits until T_b when forging the past is needed ($T_b >> T_a$)
- node forges E_k so that $E_k \leftarrow E_i$
- node announces E_k as if it were present at T_a all along
- hash function makes forging the past hard

Precomputing and Forging the Future

- to precompute an event so that it can be inserted at a later time
- node precomputes future content at T_a
- node waits until T_b when inserting the precomputed future is needed ($T_b >> T_a$)
- node inserts E_i at T_b having the precomputed content
- node announces E_i as if it were created at T_b
- digital representations of real world events contain only a small subset of the available information, the hash of a contained clock event must be used to designate the digitally represented subset (is this always possible?)
- precomputing all possible digital representations is generally hard (real world simulation)
- hash function makes precomputing only the required subset of the digital representation of the future and including in the chain hard