RFC-012: Lamport Logical Clocks

Status: Proposed **Date:** January 2026 **Author:** Derrell Piper ddp@eludom.net

---

Abstract

This RFC specifies Lamport logical clocks for Cyberspace distributed ordering, providing a happened-before relation across federated vaults without synchronized physical clocks.

Motivation

Physical clocks lie:

- **Clock skew**: Machines disagree on time
- **Clock drift**: Skew grows over time
- **NTP failures**: Synchronization breaks
- **Relativity**: No global "now" anyway

Lamport clocks provide:

1. **Logical ordering**: a $\rightarrow$ b means a happened before b
2. **No synchronization**: No NTP, no GPS, no trusted time
3. **Causality tracking**: If a caused b, then C(a) $<$ C(b)
4. **Consistency**: All nodes agree on partial order

From Lamport (1978):

> *Time, Clocks, and the Ordering of Events in a Distributed System*

Specification

The Happened-Before Relation

Definition: a $\rightarrow$ b (a happened before b) if:

1. a and b are in same process and a comes before b, or
2. a is sending a message and b is receiving it, or
3. c such that a $\rightarrow$ c and c $\rightarrow$ b (transitivity)

Lamport Clock Rules

Each process P maintains counter C(P):

```
Rule 1: Before each event, increment C(P)
         C(P) := C(P) + 1

Rule 2: When sending message m, attach C(P)
         send(m, C(P))

Rule 3: When receiving message (m, T), update clock
         C(P) := max(C(P), T) + 1
```

Clock Condition

If a $\rightarrow$ b then C(a) $<$ C(b).

Note: Converse is not guaranteed. C(a) $<$ C(b) does not imply a $\rightarrow$ b.

Data Structures

Logical Timestamp

```
(define-record-type <lamport-clock>
  (make-lamport-clock counter node-id)
  lamport-clock?
  (counter clock-counter)
  (node-id clock-node-id))
```

Timestamped Event

```
(define-record-type <timestamped-event>
  (make-timestamped-event clock event)
  timestamped-event?
  (clock event-clock)
  (event event-data))
```

Operations

Increment (Local Event)

```
(define (clock-tick! clock)
  "Increment clock for local event"
  (set! (clock-counter clock)
        (+ 1 (clock-counter clock)))
  clock)
```

Send (Message Departure)

```
(define (clock-send clock message)
  "Attach timestamp to outgoing message"
  (clock-tick! clock)
  (make-timestamped-message
    (clock-counter clock)
    (clock-node-id clock)
    message))
```

Receive (Message Arrival)

```
(define (clock-receive! clock timestamped-message)
  "Update clock on message receipt"
  (let ((remote-time (message-timestamp timestamped-message)))
    (set! (clock-counter clock)
          (+ 1 (max (clock-counter clock) remote-time)))
    (message-payload timestamped-message)))
```

Total Ordering

Lamport clocks give partial order. For total order, break ties with node ID:

```
(define (lamport-compare a b)
  "Total order: compare by timestamp, then by node-id"
  (let ((ta (clock-counter (event-clock a)))
        (tb (clock-counter (event-clock b)))
        (na (clock-node-id (event-clock a)))
        (nb (clock-node-id (event-clock b))))
    (cond
      ((< ta tb) -1)
      ((> ta tb) 1)
      ((string<? na nb) -1)
      ((string>? na nb) 1)
      (else 0))))
```

Application to Cyberspace

Audit Trail Ordering

```
(audit-entry
  (lamport-clock 1042 "alice-vault")
  (timestamp "2026-01-06T15:30:00Z")  ; Physical (advisory)
  (action (seal-commit "abc123"))
  ...)
```

Logical clock provides causal ordering even if physical clocks differ.

Federation Message Ordering

```
(federation-message
  (logical-time 573)
  (from "bob-vault")
  (type announcement)
  (payload ...))
```

Receivers update their clocks, ensuring consistent view of causality.

Conflict Detection

If events are concurrent (neither a → b nor b → a):

```
(define (concurrent? a b)
  "True if neither event happened before the other"
  (let ((ta (clock-counter (event-clock a)))
        (tb (clock-counter (event-clock b))))
    ;; If timestamps equal and different nodes, concurrent
    ;; More sophisticated: use vector clocks for accurate detection
    (and (= ta tb)
         (not (equal? (clock-node-id (event-clock a))
                      (clock-node-id (event-clock b)))))))
```

Vector Clocks (Extension)

For precise concurrency detection, use vector clocks:

```
(define-record-type <vector-clock>
  (make-vector-clock entries)
  vector-clock?
  (entries vc-entries))  ; Hash: node-id → counter

;; a → b iff VC(a) < VC(b) (component-wise)
;; a || b (concurrent) iff neither VC(a) < VC(b) nor VC(b) < VC(a)
```

Trade-off: O(N) space vs O(1) for Lamport clocks.

Integration Points

With Audit Trail (RFC-003)

```
(audit-entry
  (sequence 42)           ; Local sequence
```

```
  (lamport-clock 1042)    ; Logical timestamp
  (physical-time "...")   ; Advisory only
  ...)
```

With Consensus (RFC-011)

```
(consensus-message
  (sequence 573)          ; Consensus sequence
  (lamport-clock 2891)    ; Causal context
  ...)
```

With Replication (RFC-001)

```
(seal-publish "2.0.0"
  (logical-time 892)
  ...)
```

Security Considerations

Clock Manipulation

A Byzantine node could: - Report artificially high clock values - Attempt to order events incorrectly

Mitigations: - Clock values bounded by received values + margin - Signatures prevent retroactive tampering - Audit trails detect anomalies

Denial of Service

Flooding with high-timestamped messages forces clock advancement.

Mitigation: Rate limiting, reputation systems.

Implementation Notes

Thread Safety

Clock updates must be atomic:

```
(define (atomic-tick! clock)
  (mutex-lock! clock-mutex)
  (clock-tick! clock)
  (mutex-unlock! clock-mutex))
```

Persistence

Clock state survives restarts:

```
(define (save-clock clock path)
  (with-output-to-file path
    (lambda ()
      (write `(lamport-clock
                (counter ,(clock-counter clock))
                (node-id ,(clock-node-id clock)))))))
```

References

1. Lamport, L. (1978). Time, Clocks, and the Ordering of Events in a Distributed System.
2. Fidge, C. J. (1988). Timestamps in Message-Passing Systems.
3. Mattern, F. (1989). Virtual Time and Global States of Distributed Systems.
4. RFC-003: Cryptographic Audit Trail
5. RFC-010: Federation Protocol

Changelog

- **2026-01-06** - Initial specification

Implementation Status: Proposed **Clock Type:** Lamport scalar, vector clock extension **Ordering:** Partial (causal), total with tie-breaking