

# RFC-016: Lazy Clustering

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

---

## Abstract

This RFC specifies Lazy Clustering for Cyberspace federation: a relaxed synchronization model where nodes sync when convenient, not continuously. Optimized for the “loose confederacy of friends” model.

---

## Motivation

Not every deployment needs Byzantine consensus:

- **Friends trust friends:** No active adversary
- **Bandwidth costs:** Continuous sync is expensive
- **Offline operation:** Internet isn’t always available
- **Simplicity:** Complex protocols have complex bugs

Lazy Clustering provides:

1. **Sync when ready:** Push/pull at human pace
2. **Offline-first:** Work without network
3. **Conflict detection:** Know when divergence happens
4. **Manual resolution:** Humans resolve, not algorithms
5. **Audit everything:** Full history preserved

*The best protocol is the one you don’t run.*

---

## Specification

### Cluster Model

Alice  
(lazy)

sync when ready @ 10Gb/s

Bob Carol  
(lazy) 10Gb/s (lazy)

No heartbeats. No leader election. No quorum.  
Just friends sharing when they're ready.  
At line speed when they do.

## Performance Model

### Lazy semantics, line speed execution.

Line speed (10 Gb/s):

Typical release: 10 MB archive  
Transfer time: 8 ms  
Effective rate: ~1000 releases/second

Starlink (100-200 Mb/s, 20-40ms latency):

Typical release: 10 MB archive  
Transfer time: 400-800 ms  
Effective rate: ~1-2 releases/second  
Optimized for: Bursty, high-latency satellite links

Minimum bandwidth:

Floor: 128 Kb/s (dual-ISDN)  
Target: 1 Mb/s (T1)  
Typical release: 10 MB archive  
Transfer time: ~80 sec (T1), ~10 min (dual-ISDN)  
Strategy: Delta sync, compressed archives

### Crypto overhead (constant):

Signature verify: ~10 s (Ed25519)  
Hash verify: ~1 ms (SHA-512, 10MB)  
Total overhead: ~1 ms (negligible vs transfer)

“Lazy” means *when*, not *how fast*. When you sync, it saturates the pipe.

**Design priorities:** 1. Optimized for Starlink and satellite links 2. Tolerant of high latency (no chatty protocols) 3. Graceful degradation to minimum bandwidth 4. Bursty transfer patterns (sync then idle)

## Heartbeat and Timekeeping

No mandatory heartbeat. That’s the design.

Traditional cluster: ping → pong → ping → pong → ...

Lazy cluster: ... silence ... (sync) ... silence ...

**Timekeeping:** Lamport clocks (RFC-012), not wall clocks. – Causality without synchronization – No NTP dependency – No GPS required – Works across time zones, planets

**When you need consensus:** Byzantine consensus (RFC-011) + Lamport clocks. – Lazy clustering for everyday sync – Byzantine consensus for critical decisions – Same Lamport clock across both modes

### Optional presence beacon:

```
(cluster-beacon
  (peer "alice")
  (lamport-time 4271)
  (last-release "2.1.0")
  (status available)
  (next-expected "when ready"))
```

Beacons are: – Pull-based (query, don't push) – Cached (no flood) – Stale-tolerant (hours/days old is fine) – Unsigned (advisory only)

### Sync Modes

#### Push (I have something)

```
(lazy-push peer)
;; Sends my new releases to peer
;; Non-blocking, fire-and-forget
```

#### Pull (What do you have?)

```
(lazy-pull peer)
;; Fetches peer's new releases
;; Verifies signatures, stores locally
```

#### Sync (Bidirectional)

```
(lazy-sync peer)
;; Push then pull
;; Returns conflict report if any
```

### Lazy Semantics

**No continuous connection.** Nodes are offline by default.

**No consistency guarantees.** Nodes may diverge.

**No automatic resolution.** Conflicts flagged for humans.

**No urgency.** Sync happens when convenient.

---

## State Tracking

### Version Vector

Each node tracks what it knows about others:

```
(define-record-type <version-vector>
  (make-version-vector entries)
  version-vector?
  (entries vv-entries)) ; Hash: node-id → latest-sequence

;; Alice's view:
;; { alice: 42, bob: 37, carol: 29 }
```

### Sync Calculation

```
(define (compute-sync-set local-vv remote-vv)
  "What to send/receive"
  (let ((to-send '())
        (to-receive '()))
    (for-each
      (lambda (node)
        (let ((local-seq (vv-get local-vv node))
              (remote-seq (vv-get remote-vv node)))
          (cond
            ((> local-seq remote-seq)
             (push! to-send (releases-between node remote-seq local-seq)))
            ((< local-seq remote-seq)
             (push! to-receive (list node remote-seq local-seq))))))
      (all-nodes local-vv remote-vv))
    (values to-send to-receive)))
```

---

## Conflict Detection

### Divergence

Same version, different content:

```
(lazy-sync "bob")
;; =>
;; Conflict detected:
;; Version 2.1.0
;; Local: sha512:abc123...
;; Remote: sha512:def456...
;;
;; Both modified since common ancestor 2.0.0
```

## Conflict Record

```
(conflict
  (version "2.1.0")
  (local-hash "sha512:abc123...")
  (remote-hash "sha512:def456...")
  (common-ancestor "2.0.0")
  (detected "2026-01-06T15:30:00Z")
  (status pending)) ; pending, resolved-local, resolved-remote, merged
```

## Resolution

Manual resolution required:

```
(lazy-resolve "2.1.0" prefer: 'local)
;; or
(lazy-resolve "2.1.0" prefer: 'remote)
;; or
(lazy-resolve "2.1.0" merged: "2.1.0-merged")
```

---

## Offline Operation

### Work Offline

```
(seal-commit "Add feature")
(seal-release "2.2.0")
;; All local, no network required
```

### Queue for Sync

```
(lazy-queue)
;; =>
;; Pending sync:
;; 2.1.1 (local, not pushed)
;; 2.2.0 (local, not pushed)
```

```
;;  
;; To sync: (lazy-push "bob")
```

## Reconnect and Sync

```
(lazy-sync "bob")  
;; Sends 2.1.1, 2.2.0  
;; Receives bob's changes  
;; Reports any conflicts
```

---

## Cluster Operations

### Join Cluster

```
(lazy-join "bob"  
  uri: "git@github.com:bob/vault.git"  
  key: bob-public-key)  
;; Registers peer, doesn't sync yet
```

### Initial Sync

```
(lazy-pull "bob")  
;; Gets bob's full history  
;; Verifies all signatures
```

### Leave Cluster

```
(lazy-leave "bob")  
;; Removes peer from sync list  
;; Keeps local copies of bob's releases
```

### Cluster Status

```
(lazy-status)  
;; =>  
;; Cluster peers:  
;;   bob      last-sync: 2026-01-05  versions: 1.0.0-2.1.0  
;;   carol    last-sync: 2026-01-03  versions: 1.0.0-2.0.0 (2 behind)  
;;   dave     last-sync: never       versions: none         (not synced)  
;;  
;; Local: 2.2.0 (2 ahead of cluster)
```

---

## Sync Strategies

### Manual (Default)

```
(vault-config 'sync-strategy 'manual)
;; User explicitly calls lazy-sync
```

### Periodic

```
(vault-config 'sync-strategy 'periodic)
(vault-config 'sync-interval 3600) ; hourly
;; Background sync when network available
```

### On-Commit

```
(vault-config 'sync-strategy 'on-commit)
;; Push after each seal-commit
;; Still lazy (non-blocking, best-effort)
```

### On-Release

```
(vault-config 'sync-strategy 'on-release)
;; Push only after seal-release
;; Most conservative
```

---

## Consistency Guarantees

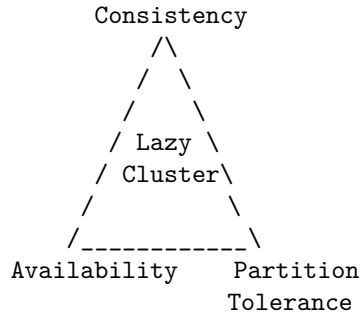
### What We Guarantee

- **Signature integrity:** All releases verified
- **Causal ordering:** Within single node
- **Conflict detection:** Divergence detected
- **Audit preservation:** Full history kept

### What We Don't Guarantee

- **Global ordering:** Nodes may see different orders
- **Consistency:** Nodes may have different content
- **Availability:** Offline nodes are offline
- **Automatic resolution:** Humans must resolve

## CAP Theorem Position



We choose AP: Available and Partition-tolerant, eventually consistent.

---

## Comparison with Other Modes

Aspect	Lazy Cluster	Federation (RFC-010)	Byzantine (RFC-011)
Trust	Friends	Verified peers	Adversarial
Sync	Manual/periodic	Announcement-based	Consensus
Conflicts	Manual resolve	Detect + flag	Prevented
Offline	Full support	Partial	Requires quorum
Complexity	Minimal	Medium	High
Use case	Small groups	Organizations	Critical systems

---

## Example Session

```
;; Morning: Alice works offline
(seal-commit "Add authentication")
(seal-commit "Add authorization")
(seal-release "2.3.0")
```

```
;; Lunch: Alice syncs with Bob
(lazy-sync "bob")
;; => Pushed 2.3.0 to bob
;; => Pulled 2.2.1 from bob
;; => No conflicts
```

```
;; Evening: Alice syncs with Carol
(lazy-sync "carol")
```



```
;; => Pushed 2.2.1, 2.3.0 to carol  
;; => Pulled nothing (carol hasn't released)  
;; => Carol has unsynced commits (not our concern)
```

---

## Security Considerations

### Trust Model

Lazy clustering assumes good-faith peers:

- Peers won't inject malicious releases
- Peers won't withhold releases maliciously
- Peers will eventually sync

**Not suitable for:** Adversarial environments, high-value targets.

**Suitable for:** Research groups, open source projects, friend networks.

### Signature Verification

All releases still verified:

```
(lazy-pull "bob")  
;; Each release:  
;; 1. Verify Ed25519 signature  
;; 2. Verify hash matches content  
;; 3. Check against known bob public key  
;; 4. Store only if valid
```

### Conflict Attacks

Malicious peer creates conflicting release.

**Mitigation:** – Conflicts flagged, not auto-resolved – Full audit trail of conflict – Peer reputation tracking

---

## Implementation Notes

### Dependencies

- crypto-ffi – Signature verification
- audit – Sync logging
- Transport (git/HTTP/filesystem)

## Storage

```
.vault/  
  lazy/  
    peers.sexp      # Registered peers  
    vectors.sexp    # Version vectors  
    conflicts/      # Unresolved conflicts  
    queue/          # Pending pushes
```

---

## References

1. Saito, Y., & Shapiro, M. (2005). Optimistic Replication.
  2. Terry, D., et al. (1995). Managing Update Conflicts in Bayou.
  3. DeCandia, G., et al. (2007). Dynamo: Amazon's Key-Value Store.
  4. RFC-010: Federation Protocol
  5. RFC-012: Lamport Logical Clocks
- 

## Changelog

- **2026-01-06** – Set minimum bandwidth: 128 Kb/s floor (dual-ISDN), 1 Mb/s target (T1)
  - **2026-01-06** – Initial specification
- 

**Implementation Status:** Proposed **Consistency Model:** Eventual (AP) **Sync Model:** Manual/periodic, offline-first **Trust Model:** Friends (non-adversarial)