Tapestry with Replication

Architecture, Algorithms, and Evaluation

Shaunak Biswas & Abhishek Sharma

November 30, 2025

Distributed Systems Project 26

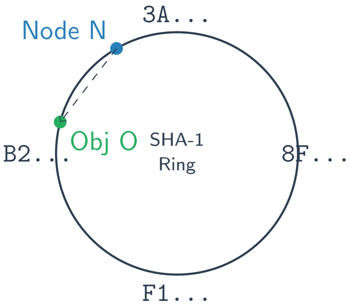
System Model & Identifier Space

Identifier Space

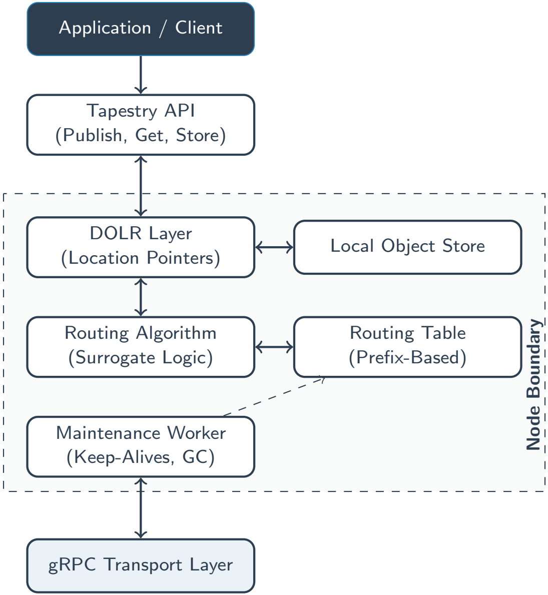
- **160-bit Space:** Circular namespace (2^{160}).
- **Representation:** 40 Hexadecimal digits.
- **Identity:** Nodes and Objects hash (SHA-1) to the same space.

Distance Metric

- Unlike Kademlia (XOR), Tapestry uses **Prefix Matching**.
- Distance = Length of shared prefix.
- *Close* = Long common prefix.



Component Interaction



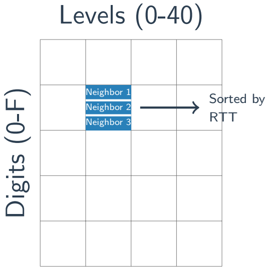
The Routing Mesh & Data Structures

Routing Table Structure

- **Dimensions:** 40 Levels × 16 Hex Digits.
- **Entry:** $Table[L][D]$ stores nodes sharing prefix length L with next digit D.

Neighbor Management

- **Redundancy:** Stores $k = 3$ backups per cell.
- **Proximity:** Neighbors sorted by RTT (Latency).
- **Backpointers:** Inverse graph tracking nodes that point to *me* (Critical for Leave/Join).



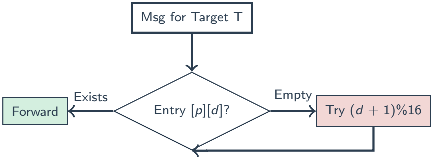
Routing Logic: Surrogate Routing

Next Hop Selection

1. Calculate shared prefix length p.
2. Look up digit d at level $p + 1$.
3. **Match:** If entry exists, forward to closest neighbor.
4. **Hole:** If entry empty, invoke **Surrogate Routing**.

Surrogate Mechanism

Deterministic modulo search: $(d + 1, d + 2, \dots) \pmod{16}$. Guarantees convergence to a unique root even if target ID doesn't exist.



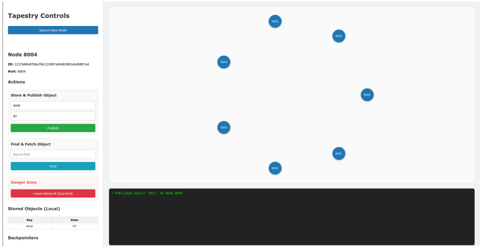
Decentralized Object Location (DOLR)

Publication (Advertisement)

- **Hash:** Key hashed to SHA-1 ObjectID.
- **Route:** Message routes toward Root.
- **Cache:** Intermediate nodes store *Location Pointers* (Soft State).
- Mapping: $ObjectID \rightarrow PublisherAddr$.

Lookup (Resolution)

- Route toward Root.
- **Interception:** If any node has a pointer, return it immediately.
- Reduces latency by finding cached pointers closer to the client.



Tapestry Controls

Node 8004

ID: 1237b08a0fb8af6b122887a84d01801ded0807ad

Port: 8004

Actions

Store & Publish Object

Amit

97

Find & Fetch Object

Danger Zone

Stored Objects (Local)

Key	Data
Amit	97

Backpointers

```
> Published object 'Amit' on Node 8004
```

Visualization: Nodes storing pointers along the path

Dual Replication Strategy

1. **Storage Replication (Durability)**
   - **Goal:** Survive Publisher crash.
   - **Action:** On Publish, replicate data payload to N random neighbors.
   - **Protocol:** Custom `Replicate` RPC.

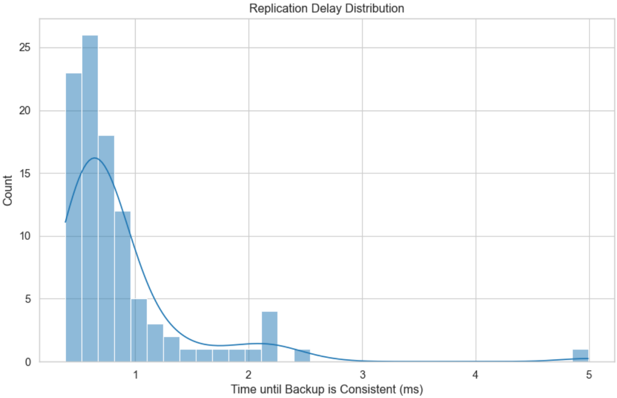
2. **Salted Path Replication (Availability)**
   - **Goal:** Survive Root failure / Partition.
   - **Action:** Publish to multiple IDs: $H(K), H(K + "0"), H(K + "1")$.
   - Creates independent paths to distinct roots.

Replication Logic

This dual strategy separates Data Durability (storing copies) from Routing Availability (finding copies).

Replication Performance

Replication Delay Distribution



Most replication operations complete within 3ms, minimizing the window of vulnerability.

Soft State Maintenance

The network assumes all information will eventually become stale.

Republishing Loop

- **Who:** Publisher Nodes.
- **Action:** Re-advertise objects every 60s.
- **Why:** Refreshes pointers at routers.

Garbage Collection (GC)

- **Who:** Routers (Intermediate nodes).
- **Action:** Scan `LocationPointers`.
- **Logic:** Delete entries older than 120s.
- **Result:** Dead paths are automatically pruned.

Keep-Alives: Background probes remove unresponsive neighbors from the Routing Table.

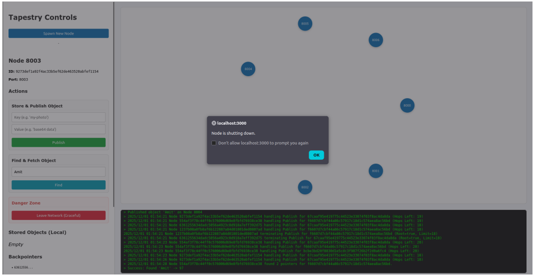
Node Lifecycle: Join & Leave

Insertion (Join)

1. **Bootstrap:** Connect to gateway.
2. **Route to Self:** Find Surrogate node.
3. **Copy:** `GetRoutingTable` from Surrogate.
4. **Notify:** `AddBackpointer` to neighbors.

Graceful Departure (Leave)

1. **Handoff:** Push local objects to neighbors via `Replicate`.
2. **Notify:** Send `NotifyLeave` to Backpointers.
3. Peers remove node immediately.



Node 8003

ID: 9273def1a92f4ac33b5ef62de463520abfef1154

Port: 8003

Amit

Stored Objects (Local)

Empty

Backpointers

- 63612556...

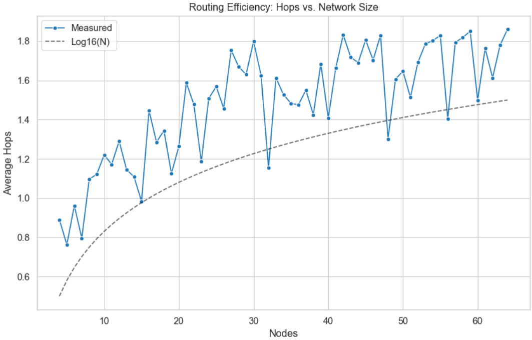
localhost:3000

Node is shutting down.

□ Don't allow localhost:3000 to prompt you again

```
> Published object 'Amit' on Node 8004
> 2025/12/01 01:54:21 Node 9273def1a92f4ac33b5ef62de463520abfef1154 handling Publish for 67caaf95e419775c44523e33074f03f8ac4da8da (Hops Left: 19)
> 2025/12/01 01:54:21 Node 554af3f78c44ff0c576006d69e8fbfd76938ce38 handling Publish for f6607d7cbf44a86c57917c18d1c574aea8ac56bd (Hops Left: 19)
> 2025/12/01 01:54:21 Node 6361255634dadc5056a4923c0d918a7eff392d75 handling Publish for 67caaf95e419775c44523e33074f03f8ac4da8da (Hops Left: 18)
> 2025/12/01 01:54:21 Node 1237b08a0fb8af6b122887a84d01801ded0807ad handling Publish for f6607d7cbf44a86c57917c18d1c574aea8ac56bd (Hops Left: 18)
2025/12/01 01:54:21 Node 1237b08a0fb8af6b122887a84d01801ded0807ad terminating Publish for f6607d7cbf44a86c57917c18d1c574aea8ac56bd (Root=true, Limit=18)
> 2025/12/01 01:54:21 Node 6361255634dadc5056a4923c0d918a7eff392d75 terminating Publish for 67caaf95e419775c44523e33074f03f8ac4da8da (Root=true, Limit=18)
> 2025/12/01 01:54:23 Node 554af3f78c44ff0c576006d69e8fbfd76938ce38 handling Publish for 67caaf95e419775c44523e33074f03f8ac4da8da (Hops Left: 20)
2025/12/01 01:54:23 Node 554af3f78c44ff0c576006d69e8fbfd76938ce38 handling Publish for f6607d7cbf44a86c57917c18d1c574aea8ac56bd (Hops Left: 20)
2025/12/01 01:54:23 Node 554af3f78c44ff0c576006d69e8fbfd76938ce38 handling Publish for bcbe3bd20f8830d1e5cdc3f987f26b2f2d0a6fc4 (Hops Left: 20)
> 2025/12/01 01:54:23 Node 9273def1a92f4ac33b5ef62de463520abfef1154 handling Publish for 67caaf95e419775c44523e33074f03f8ac4da8da (Hops Left: 18)
> 2025/12/01 01:54:26 Node 9273def1a92f4ac33b5ef62de463520abfef1154 handling Publish for 67caaf95e419775c44523e33074f03f8ac4da8da (Hops Left: 18)
> 2025/12/01 01:54:28 Node 554af3f78c44ff0c576006d69e8fbfd76938ce38 found 2 pointers for f6607d7cbf44a86c57917c18d1c574aea8ac56bd.
> Success: Found 'Amit' -> 97
```

Evaluation: Routing Efficiency



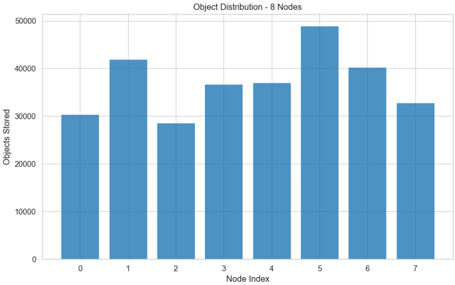
Scaling

Hop count scales logarithmically $O(\log_{16} N)$.

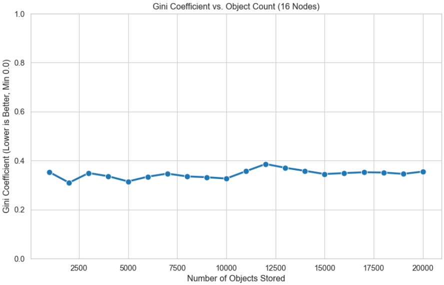
The aggressive table population during Join keeps hop counts close to theoretical bounds ($\approx$ 1.6 hops).

Evaluation: Load Distribution

Object Histogram



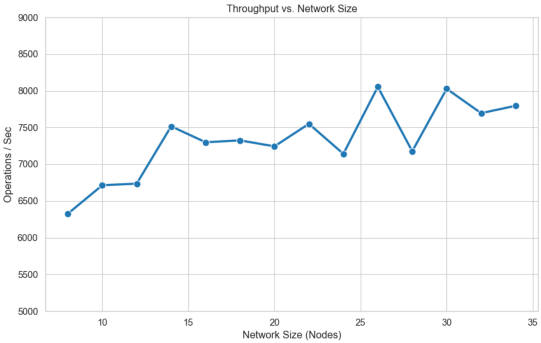
Gini Coefficient



Insight

Low Gini coefficient confirms SHA-1 hashing distributes objects uniformly, preventing hotspots even with random keys.

Evaluation: System Throughput

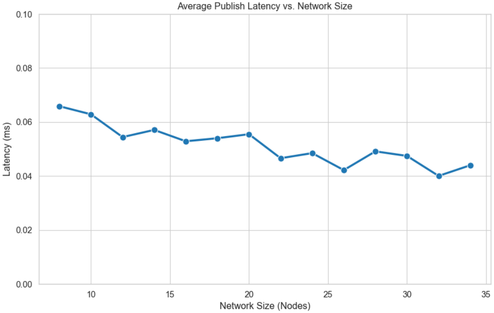


Stability

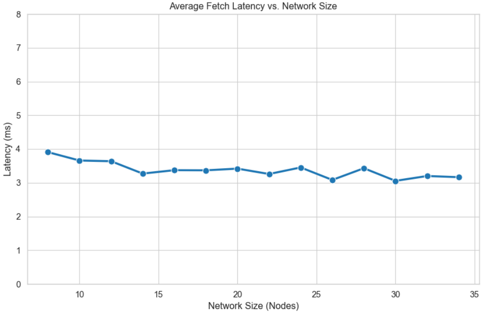
Throughput increases in larger networks. This proves that our **Connection Pooling** successfully mitigated OS resource exhaustion (ephemeral ports).

Evaluation: Latency (Publish vs Fetch)

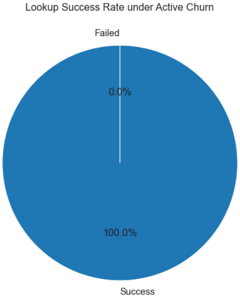
Publish Latency



Fetch Latency



Evaluation: Resilience to Churn



Stress Test

Scenario: Randomized Node Failure.

Result: High Success Rate. Salted paths and Storage Replication successfully masked node failures during the test window.

Conclusion

We implemented a full-featured Tapestry overlay in Go, verifying the original research claims while adding modern engineering robustness.

- ✓ **Core Protocol:** Prefix Routing, Surrogate Logic, Soft State.
- ✓ **Replication and Fault Tolerance:** Salted Paths + 3-Way Storage Replication.
- ✓ **Lifecycle:** Graceful Handoff and Fast Join optimizations.
- ✓ **Performance:** Validated Logarithmic scaling and High Availability under churn.

Thank You