Tapestry
Architecture and status

UCB ROC / Sahara Retreat
January 2002

Ben Y. Zhao
ravenben@eecs.berkeley.edu

What is Tapestry?

- A prototype of a *decentralized, fault-tolerant, adaptive* overlay infrastructure
  (Zhao, Kubiatowicz, Joseph et al. 2000)
- Network substrate of OceanStore
  - Routing: Suffix-based hypercube
    Similar to Plaxton, Rajamaran, Richa (SPAA97)
  - Decentralized location:
    Virtual hierarchy per object with cached location references
- Dynamic algorithms using local information
- Core API:
  - *publishObject(ObjectID)*
  - *routeMsgToObject(ObjectID)*
  - *routeMsgToNode(NodeID)*

Why Tapestry

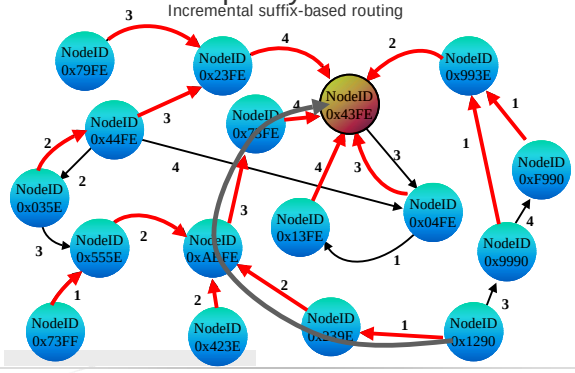
- Today's Internet
  - Route failures not uncommon
    - BGP too slow to recover, redundant routes unexploited
  - IPv4 constrains deployment of new protocols
    - IP multicast, security protocols (DDoS traceback), …
  - Wide-area applications straining existing systems
    - Scalable management of large scale resources
- Our goals
  - Wide-area scalable network overlay
    - Highly fault-tolerant routing / location
    - Introspective / self-tuning platform
    - Support application-specific protocols
    - Efficient (b/w, latency) data delivery
  - Pass on wide-area solutions to application layer

Routing and Location

- Namespace (nodes and objects)
  - 160 bits length → 2^{80} names before name collision
  - Each object has its own hierarchy rooted at *Root*
    f (ObjectID) = RootID, via a dynamic mapping function
- Suffix routing from A to B
  - At h^{th} hop, arrive at nearest node hop(h) such that:
    hop(h) shares suffix with B of length *h* digits
  - Example: 5324 routes to 0629 via
    5324 → 2349 → 1429 → 7629 → 0629
- Object location:
  - Root responsible for storing object's location
  - Publish / search both route incrementally to root

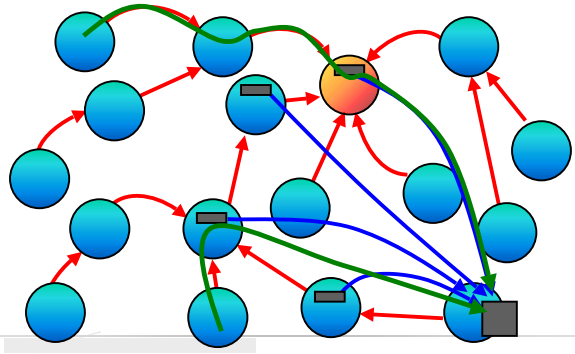
Tapestry Mesh

Incremental suffix-based routing



Object Location

Randomization *and* Locality



Fault-tolerant Routing

- Strategy:
  - Detect failures via soft-state probe packets
  - Route around problematic hop via backup pointers
- Handling:
  - 3 forward pointers per outgoing route (2 backups)
  - 2nd chance algorithm for intermittent failures
  - Upgrade backup pointers and replace
- Protocols:
  - First Reachable Link Selection (FRLS)
  - Proactive Duplicate Packet Routing

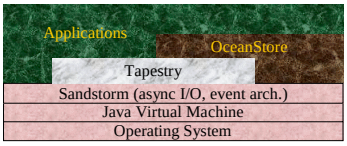
Talk Outline

- Tapestry overview
- Architecture
- Evaluation
- Brocade
- Conclude

Architecture Background

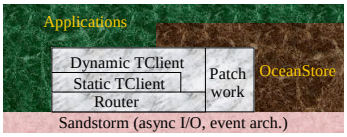
OceanStore implementation

- Java with asynchronous I/O
- Event-based, stage driven architecture (Sandstorm – M. Welsh)



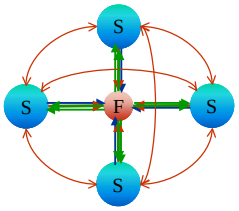
Key Stages

- StaticTClient / Federation
  - Uses config files to bootstrap initial Tapestry
- DynamicTClient
  - Integrates new nodes into static Tapestry
- Router
  - Primary handler of routing and location
- Patchwork
  - Introspective monitoring and fault-detection



Static TClient

- Federation used as rendezvous point
- Pair-wise pings to generate route tables
- Federation used as global barrier to begin

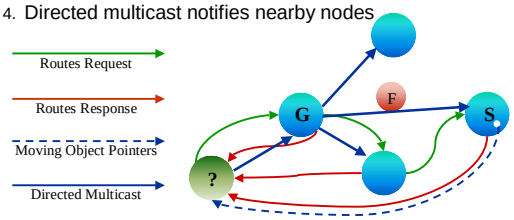


1. S_i says *hello* to F
2. F informs group of S_i
3. Nodes do pair-wise pings
4. Nodes signal readiness
5. Barrier reached at F, signals start

Dynamic TClient

Node Integration

1. Hill-climb to find nearest Gateway
2. Route to surrogate / copy routes
3. Move relevant objects to new root
4. Directed multicast notifies nearby nodes

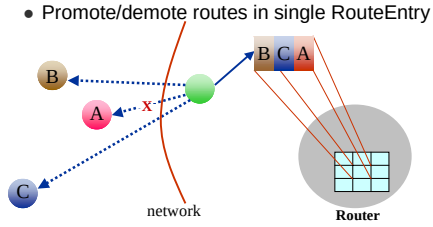


Routing / Location

- *Router* class
- Maintains:
  - RoutingTable:
    [ ][ ] of RouteEntries
  - ObjectPointers:
    Hash(Guid)→PublishInfo
    Hash(Guid)→LastHop
- Handles:
  - Object publication / unpublication / mobile objects
  - Route / location message handling

Patchwork

- Fault-handling / introspective stage
  - Granulated periodic beacons measure loss and network latency to entries in routing table
  - Promote/demote routes in single RouteEntry



Deployment Status

- ❖ Object Location
  - ✓ Publish / unpublish / route to object
  - ✓ Mobile objects (backtracking unpublish)
  - ❑ Active deletes, confirmation of non-existence
- ❖ General Routing
  - ✓ Route to node, redundant routes
  - ✓ Soft-state fault-detection, limited optimization
  - ❑ Advanced policies for fault recovery
- ❖ Dynamic Integration
  - ✓ Integration w/ limited optimizations
  - ❑ Best effort fault-resilient integration mechanisms
- ❑ Background threads for optimization / refresh

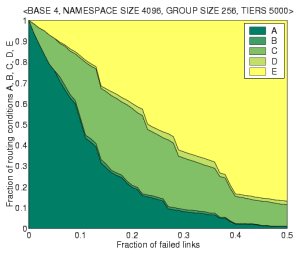
Talk Outline

- Tapestry overview
- Architecture
- Evaluation
- Brocade
- Conclude

Generalized Results

- Cached object pointers
  - Efficient lookup for nearby objects
  - Reasonable storage overhead
- Multiple object roots
  - Improves availability under attack
  - Improves performance and perf. stability
- Reliable packet delivery
  - Redundant pointers approximate optimal reachability
  - FRLS, a simple fault-tolerant UDP protocol

First Reachable Link Selection



- Use periodic UDP packets to gauge link condition
- Packets routed to shortest "good" link
- Assumes IP cannot correct routing table in time for packet delivery

	IP	Tapestry
A	✓	✓
B	✓	✗
C	✗	✓
D	✗	✗
E	No path exists to dest.	

Some Numbers

- Measurements
  - PIII 800, L2.2.18, IBM JDK 1.3
  - Simulating 6 nodes (4 staticTC, 1 federation, 1 dynamicTC)
  - Publishing / locating ~10 objects
  - PublishMsg, RouteMsg: ~ 0-2 ms
  - Integration: ~2600ms (w/ pings)
- Integration messages:
  - Assuming latency data available
  - 2 x n (routing and objects) 16^M (directed multicast notification) ($M \approx 3$)

Talk Outline

- Tapestry overview
- Architecture
- Evaluation
- Brocade
- Conclude

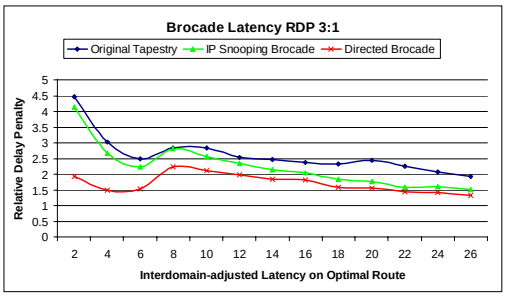
Landmark Routing on P2P

- Brocade
  - Exploit non-uniformity
  - Minimize wide-area routing hops / bandwidth
- Secondary overlay on top of Tapestry
  - Select *super*-nodes by admin. domain
    - Divide network into *cover sets*
  - Super-nodes form secondary Tapestry
    - Advertise cover set as local objects
  - Routing (A→B) uses brocade to route directly into B's local network

Brocade Mechanisms

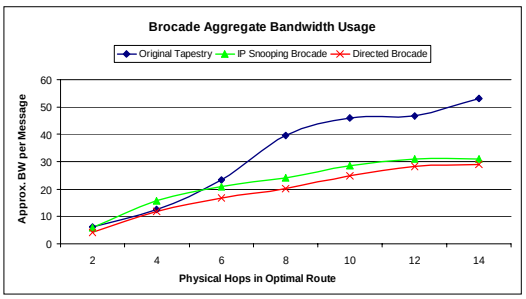
- Selective utilization
  - Nodes cache local cover set
  - Only utilize brocade if dest. not in cache
- Forwarding messages to supernodes
  1. Super-node does *IP-snooping*
  2. *Direct*: cover set caches supernode
- Inter-domain routing: A→B
  1. A→SN(A) via IP
  2. SN(A) finds SN(B) via Tapestry location
  3. SN(B)→B via Tapestry/Chord/Pastry/CAN

Brocade Routing RDP



Local cover set cache on; interdomain:intradomain = 3:1
Packet simulator, Transit-stub 4096 T nodes, 16 SuperN

Brocade Bandwidth Usage



Local cover set cache on
B/W unit: (sizeof (Msg) * Hops)

Ongoing / Future Work

- Fill in full functionality
  - Fault-handling policies, introspection, self-repair
- More realistic experiments
  - Artificial topologies on SOSS simulator
  - Larger scale dynamic integration experiments
- Code development
  - External deployment / Code release
    - Sprint programmable routers
    - Academic networks
  - Introspective measurement platform
  - Implementing applications (Bayeux, Brocade …)

For More Information

Tapestry and related projects (and these slides):
http://www.cs.berkeley.edu/~ravenben/tapestry

OceanStore:
http://oceanstore.cs.berkeley.edu

Related papers:
http://oceanstore.cs.berkeley.edu/publications
http://www.cs.berkeley.edu/~ravenben/publications

ravenben@eecs.berkeley.edu