Tapestry: Scalable and Fault-tolerant Routing and Location

Stanford Networking Seminar
October 2001

Ben Y. Zhao
ravenben@eecs.berkeley.edu

Challenges in the Wide-area

- Trends:
  - Exponential growth in CPU, storage
  - Network expanding in reach and b/w
- Can applications leverage new resources?
  - Scalability: increasing users, requests, traffic
  - Resilience: more components → inversely low MTBF
  - Management: intermittent resource availability → complex management schemes
- Proposal: an infrastructure that solves these issues and passes benefits onto applications

Driving Applications

- Leverage of cheap & plentiful resources: CPU cycles, storage, network bandwidth
- Global applications share distributed resources
  - Shared computation:
    - SETI, Entropia
  - Shared storage
    - OceanStore, Gnutella, Scale-8
  - Shared bandwidth
    - Application-level multicast, content distribution networks

Key: Location and Routing

- Hard problem:
  - Locating and messaging to resources and data
- Goals for a wide-area overlay infrastructure
  - Easy to deploy
  - Scalable to millions of nodes, billions of objects
  - Available in presence of routine faults
  - Self-configuring, adaptive to network changes
  - Localize effects of operations/failures

Talk Outline

- Motivation
- Tapestry overview
- Fault-tolerant operation
- Deployment / evaluation
- Related / ongoing work

What is Tapestry?

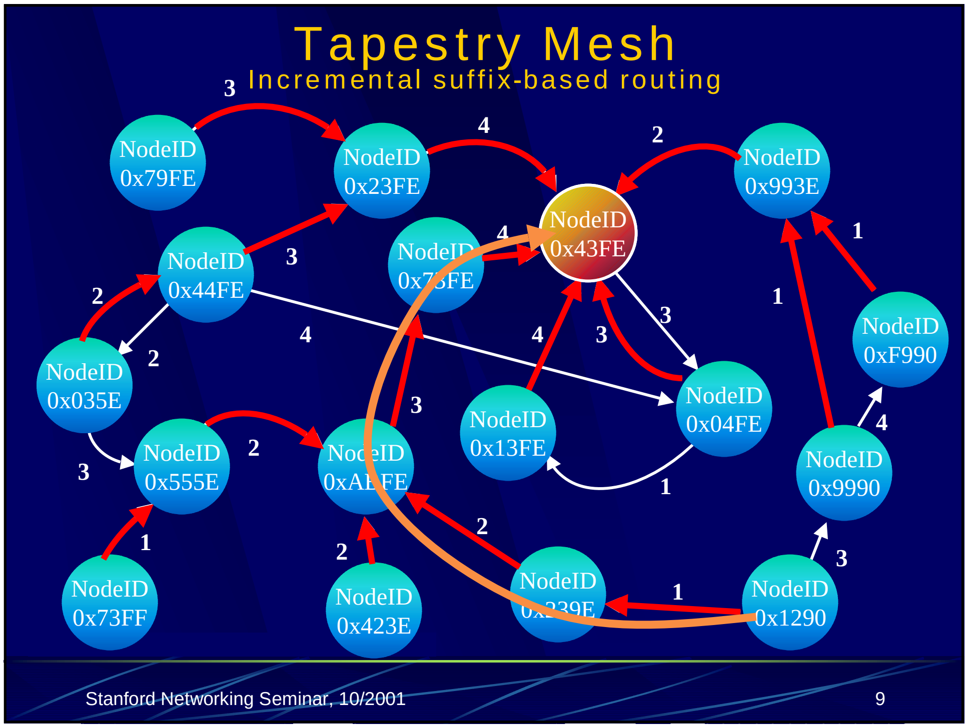
- A prototype of a *decentralized, scalable, fault-tolerant, adaptive* location and routing infrastructure *(Zhao, Kubiatowicz, Joseph et al. U.C. Berkeley)*
- Network layer of OceanStore
- Routing: Suffix-based hypercube
  - Similar to Plaxton, Rajamaran, Richa (SPAA97)
- Decentralized location:
  - Virtual hierarchy per object with cached location references
- Core API:
  - *publishObject(ObjectID, [serverID])*
  - *routeMsgToObject(ObjectID)*
  - *routeMsgToNode(NodeID)*

Routing and Location

- Namespace (nodes and objects)
  - 160 bits → 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) s.t.
    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

Publish / Lookup

- Publish object with ObjectID:
  // route towards "virtual root," ID=ObjectID
  For (i=0, i<Log_2(N), i+=j) { //Define hierarchy
  - j is # of bits in digit size, (i.e. for hex digits, j = 4)
  - Insert entry into nearest node that matches on last i bits
  - If no matches found, deterministically choose alternative
  - Found real root node, when no external routes left
- Lookup object
  Traverse same path to root as publish, except search for entry at each node
  For (i=0, i<Log_2(N), i+=j) {
  - Search for cached object location
  - Once found, route via IP or Tapestry to object



Routing in Detail

Example: Octal digits, 2^{12} namespace, 5712 → 7510

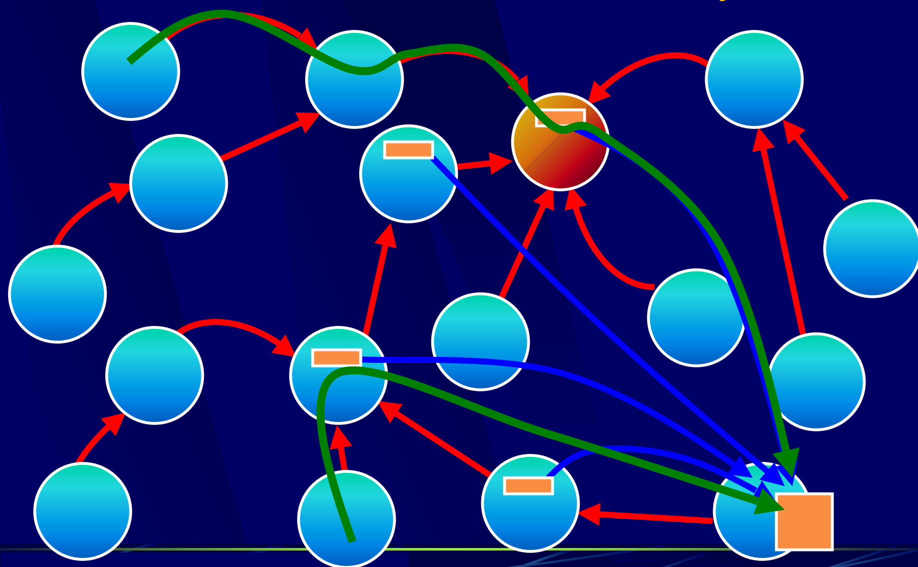
5712	0	1	2	3	4	5	6	7
0880	0	1	2	3	4	5	6	7
3210	0	1	2	3	4	5	6	7
4510	0	1	2	3	4	5	6	7
7510	0	1	2	3	4	5	6	7

Neighbor Map
For "5712" (Octal)

4	3	2	1
0712	x012	xx02	xxx0
1712	x112	5712	xxx1
2712	x212	xx22	5712
3712	x312	xx32	xxx3
4712	x412	xx42	xxx4
5712	x512	xx52	xxx5
6712	x612	xx62	xxx6
7712	5712	xx72	xxx7

Routing Levels

Object Location
Randomization *and* Locality



Talk Outline

- Motivation
- Tapestry overview
- Fault-tolerant operation
- Deployment / evaluation
- Related / ongoing work

Fault-tolerant Location

- Minimized soft-state vs. explicit fault-recovery
- Redundant roots
  - Object names hashed w/ small salts → multiple names/roots
  - Queries and publishing utilize all roots in parallel
  - P(finding reference w/ partition) = $1 - (1/2)^n$ where n = # of roots
- Soft-state periodic republish
  - 50 **million** files/node, daily republish, $b = 16$, $N = 2^{160}$, 40B/msg, worst case update traffic: 156 kb/s,
  - expected traffic w/ 2^{40} real nodes: 39 kb/s

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

Summary

- Decentralized location and routing infrastructure
  - Core routing similar to PRR97
  - Distributed algorithms for object-root mapping, node insertion / deletion
  - Fault-handling with redundancy, soft-state beacons, self-repair
  - Decentralized and scalable, with locality
- Analytical properties
  - Per node routing table size: $bLog_b(N)$
    - N = size of namespace, n = # of physical nodes
  - Find object in $Log_b(n)$ overlay hops

Talk Outline

- Motivation
- Tapestry overview
- Fault-tolerant operation
- Deployment / evaluation
- Related / ongoing work

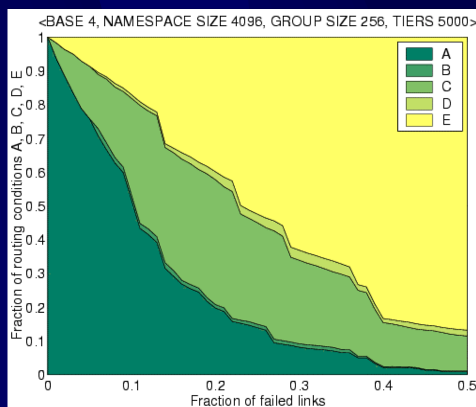
Deployment Status

- Java Implementation in OceanStore
  - Running static Tapestry
  - Deploying dynamic Tapestry with fault-tolerant routing
- Packet-level simulator
  - Delay measured in network hops
  - No cross traffic or queuing delays
  - Topologies: AS, MBone, GT-ITM, TIERS
- ns2 simulations

Evaluation 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.	

Talk Outline

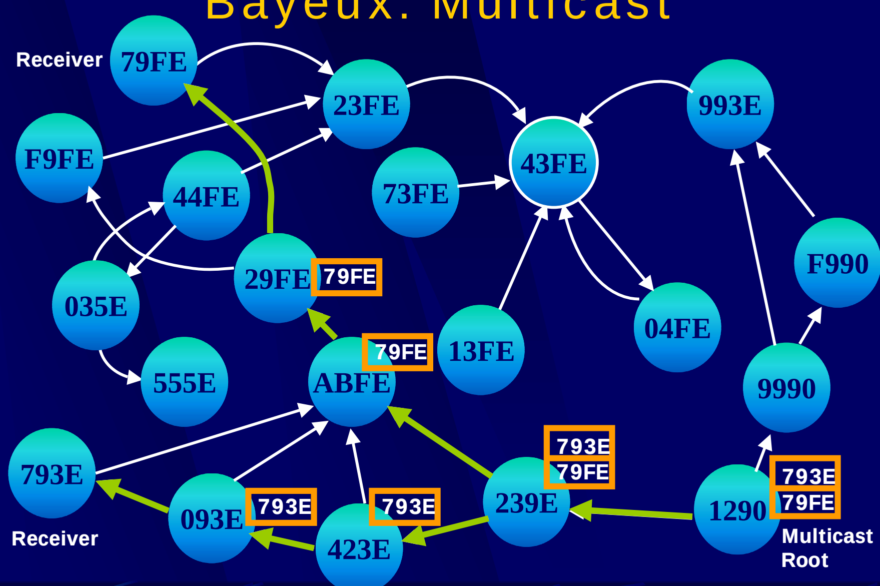
- Motivation
- Tapestry overview
- Fault-tolerant operation
- Deployment / evaluation
- Related / ongoing work

Bayeux

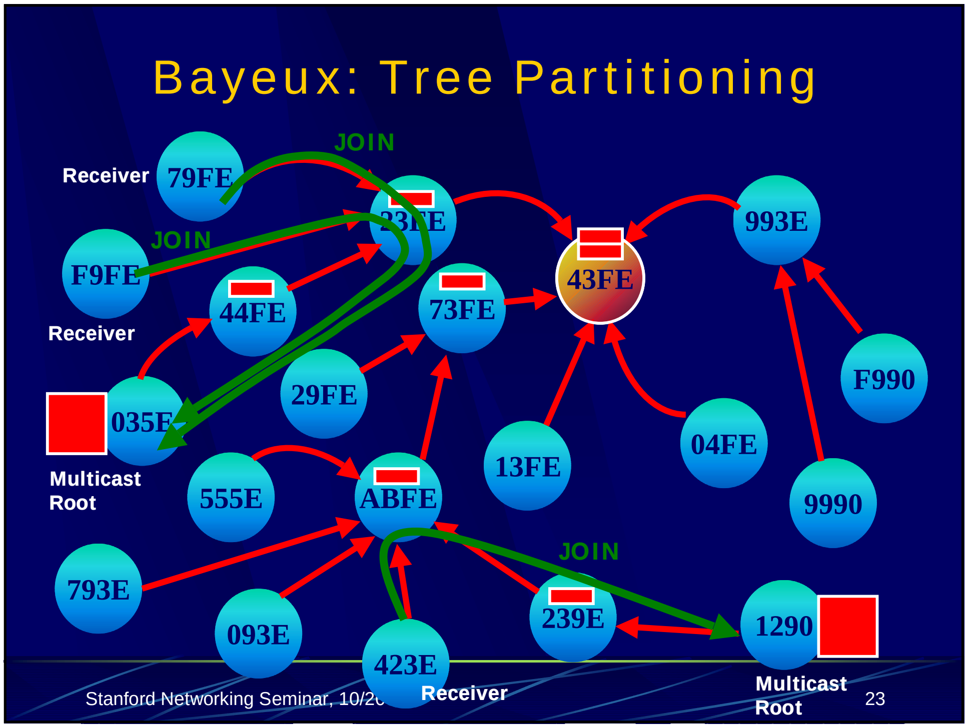
- Global-scale application-level multicast (NOSSDAV 2001)
- Scalability
  - Scales to > 10^5 nodes
  - Self-forming member group partitions
- Fault tolerance
  - Multicast root replication
  - FRLS for resilient packet delivery
- More optimizations
  - Group ID clustering for better b/w utilization

Stanford Networking Seminar, 10/2001 21

Bayeux: Multicast



Stanford Networking Seminar, 10/2001 22



Overlay Routing Networks

- **CAN:** Ratnasamy et al., (ACIRI / UCB)
  - Uses *d*-dimensional coordinate space to implement distributed hash table
  - Route to neighbor closest to destination coordinate

  Fast Insertion / Deletion
  Constant-sized routing state
  Unconstrained # of hops
  Overlay distance not prop. to physical distance

- **Chord:** Stoica, Morris, Karger, et al., (MIT / UCB)
  - Linear namespace modeled as circular address space
  - "Finger-table" point to logarithmic # of inc. remote hosts

  Simplicity in algorithms
  Fast fault-recovery
  $Log_2(N)$ hops and routing state
  Overlay distance not prop. to physical distance

- **Pastry:** Rowstron and Druschel (Microsoft / Rice)
  - Hypercube routing similar to PRR97
  - Objects replicated to servers by name

  Fast fault-recovery
  Log(N) hops and routing state
  Data replication required for fault-tolerance

Ongoing Research

- Fault-tolerant routing
  - Reliable Overlay Networks (MIT)
  - Fault-tolerant Overlay Routing (UCB)
- Application-level multicast
  - Bayeux (UCB), CAN (AT&T), Scribe and Herald (Microsoft)
- File systems
  - OceanStore (UCB)
  - PAST (Microsoft / Rice)
  - Cooperative File System (MIT)

For More Information

Tapestry:

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@cs.berkeley.edu