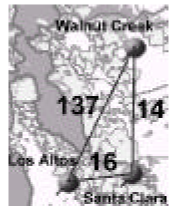
Overlay Networks

Arvind Krishnamurthy
Fall 2003

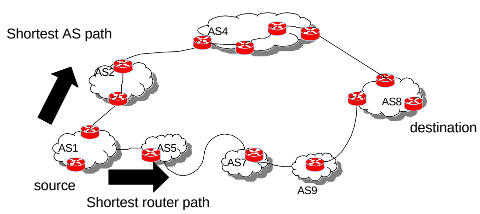
Internet Routing

- Internet routing is inefficient:
  - Does not always pick the lowest latency paths
  - Does not always pick paths with low drop rates
- Experimental evidence with 43 nodes: (Detour project at Washington)
  - 50% of routes had a faster alternate one-hop route
  - 80% of routes had an alternate route with lower loss rate



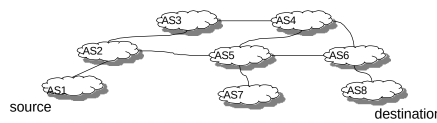
Reasons for path inflation #1

- Using AS-hop-count as routing metric:
  - Using actual latency or router-hop-count would be better



Reasons for path inflation #2

- Policy routing might result in picking a longer path
- No-valley routing policy:
  - An AS does not provide transit between any two of its providers or peers.



- Prefer Customer routing policy:
  - Prefer the free of charge customer route over the peer or provider route.
- Early exit strategy:
  - AS might try to get rid of a packet as soon as possible and minimize its intra-domain traffic

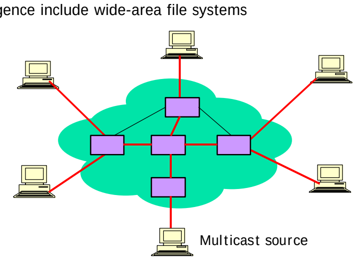
Internet Path Outages

- When a path fails try to find an alternate path if possible
- Internet:
  - Path outages are reasonably common
  - Recovery time could be substantial

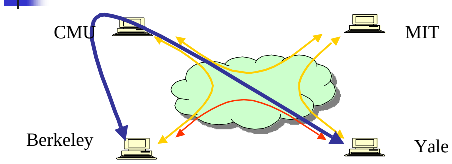
Paxson 95-97	▪ 3.3% of all routes had serious problems
Labovitz 97-00	▪ 10% of routes available < 95% of the time ▪ 65% of routes available < 99.9% of the time ▪ 3-min minimum detection+recovery time; often 15 mins ▪ 40% of outages took 30+ mins to repair
Chandra 01	▪ 5% of faults last more than 2.75 hours

Advanced Routing Mechanisms

- Internet routers rarely support advanced protocols such as IP multicast
- Ideally, routers should have intelligence to form multicast trees, maintain membership information, and split flows
- More advanced applications that could benefit from network-embedded intelligence include wide-area file systems

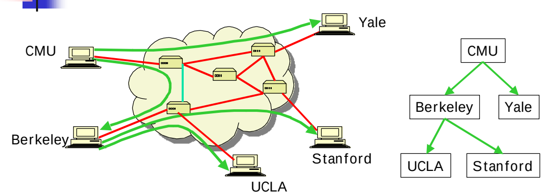


Solution: Overlay Networks



- Route around faults
  - Use an “overlay path” that comprises of two or more physical connections through the internet
- Route around inefficiencies
- Intelligence (for multicasting, wide-area file-systems, etc.) is pushed to the edge of the internet

Overlay Multicast



- Multicast performed through multiple unicasts
- Edge-hosts serve as forwarding agents (results in a distribution tree)
- Question: what constitutes a good distribution tree?
  - How would you design such a overlay multicast system?

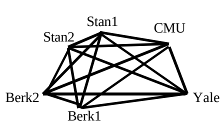
Narada Project at CMU

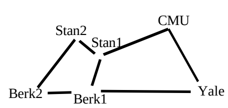
Step 0

Maintain a complete overlay graph of all group members
- Links correspond to unicast paths
- Link costs maintained by polling

Step 1

“Mesh”: Subset of complete graph may have cycles and includes all group members
- Members have low degrees
- Shortest path delay between any pair of members along mesh is small

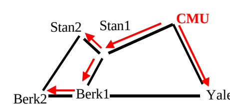




Narada Design (contd.)

Step 2

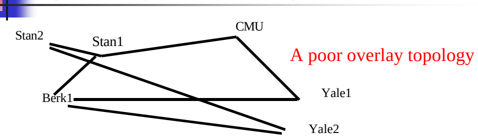
- Source rooted shortest delay spanning trees of mesh
- Constructed using distance vector routing
  - Members have low degrees
  - Small delay from source to receivers



Narada Components

- Mesh Management:
  - Ensures mesh remains connected in face of membership changes
- Mesh Optimization:
  - Distributed heuristics for ensuring shortest path delay between members along the mesh is small
- Spanning tree construction:
  - Routing algorithms for constructing data-delivery trees
  - Distance vector routing, and reverse path forwarding

Optimizing Mesh Quality



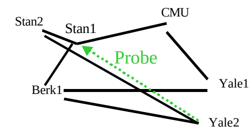
- Members periodically probe other members at random
- New Link added if
  Utility Gain of adding link > Add Threshold
- Members periodically monitor existing links
- Existing Link dropped if
  Cost of dropping link < Drop Threshold

The terms defined

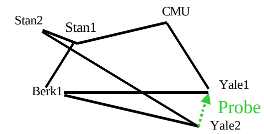
- Utility gain of adding a link based on
  - The number of members to which routing delay improves
  - How significant the improvement in delay to each member is
- Cost of dropping a link based on
  - The number of members to which routing delay increases, for either neighbor
- Add/Drop Thresholds are functions of:
  - Member's estimation of group size
  - Current and maximum degree of member in the mesh

Desirable properties of heuristics

- Stability: A dropped link will not be immediately added
- Partition Avoidance: A partition of the mesh is unlikely to be caused as a result of any single link being dropped

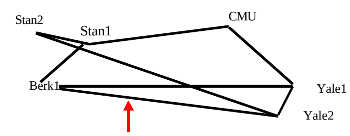


Delay improves to Stan1, CMU but marginally.
Do not add link!

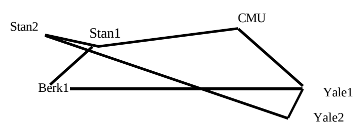


Delay improves to CMU, Yale1 and significantly.
Add link!

Mesh Improvement



Used by Berk1 to reach only Yale2 and vice versa.
Drop



An improved mesh

Narada Mesh Construction

- Discussion on metrics:
  - Option 1: use latency as the link weight metric
  - Option 2: use bandwidth (or inverse-bandwidth) as link weight
  - Option 3: use some function of latency and bandwidth (such as cost of sending 8K data)
- Hybrid metrics:
  - For some applications, have bandwidth as the important metric and use latency to choose between links that have similar bandwidth
    - Transform bandwidths into coarse levels
    - For paths/links with bandwidth at a certain coarse level, use latency as the discriminating factor

Mesh Applications

- Narada is used for end-system multicast
- Resilient Overlay Networks (RON) uses a complete graph as the mesh
  - Used for finding backup overlay paths when direct connections fail
  - Routing protocol overheads dominate
  - RON uses link state protocol – similar overheads for distance vector
  - Scales only to about 50 or so nodes
- Overlay network can also be used for multipath routing
  - Can increase net bandwidth
  - Multi-tree routing can be used to improve multicast performance
  - Can send redundant data to improve loss-rates for real-time applications

Mesh Construction Strategies

- Narada:
  - Start with random graph
  - Perform mesh improvement
- Another strategy:
  - Start with “k” good links and “k” random links per node
  - “k” random links ensures that initial mesh is connected
  - Perform mesh improvement
- Can we do better? Would like a mesh that is:
  - “k” connected (to support path redundancy)
  - Contains the best physical connections
  - Has low degree on each node
  - Has low diameter for the entire mesh

Graph Theoretic Results

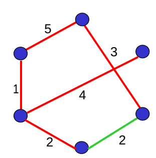
- Problem: Find a subgraph that is "k"-connected and minimizes total edge weight of edges included in the subgraph
  - NP-Hard problem for k > 1
  - k = 1, solution is minimum spanning tree
  - k = 2, becomes NP-Hard because you can reduce Hamiltonian path problem to this problem
- Best approximation algorithms:
  - Approximate by a factor of 2 -- an algorithm by Gabow that:
    - Converts original graph into a directed graph
    - Finds "k" directed trees of minimum cumulative weight
  - Problems with the approach:
    - Complex algorithm uses matroid graph theory
    - Does not adapt to distributed implementation

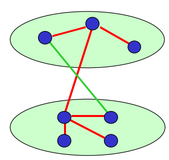
Approximate the Approximation

- Stick with the general idea of finding a subgraph comprising of "k" trees
  - Find the trees in a greedy manner
  - Find the minimum-spanning tree (MST) of the graph
  - Remove the edges, find the second MST
  - Repeat the process k times
  - Mesh comprises of the k-MSTs

k-MST Properties

- For each node, it includes the "k" best links coming out of the node
- If a new link with lower weight is "found":
  - try to add the link
  - creates a cycle
  - delete the link with the highest weight along the cycle
- If an existing link becomes more expensive:
  - Let S1 and S2 be the sets connected by the link
  - Find the link connecting S1 and S2 in the next higher tree
  - Replace existing link with the link from higher tree





Distributed algorithm

- Extend the Gallager-Humblet-Spira algorithm for finding Minimum Spanning tree
- Impose degree and diameter heuristics
  - When two components merge in GHS, accept/reject the merge based on current degree bound
  - Also reject merge if it happens at the "outer ends" of the components
- Setup a pipeline of "k" GHS computations
  - Edge rejected in an earlier computation is passed to a later computation
- When a node detects a lower weight link, it tries to "lock" the links by sending a message along the associated fundamental cycle
  - Once the links along the cycle are locked, it can swap in the new link for an old link

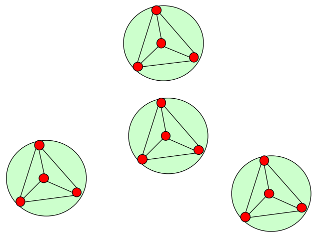
Mesh-based approaches

- Require a routing algorithm eventually
  - To find alternate shortest paths
  - Or find a distribution tree
- Overheads associated with routing algorithms limit scalability
  - Fully connected graph (as in RON) scales to only 50-100 nodes
  - Narada/k-MST approaches scale to about 1000 nodes
- Can we devise "implicit" overlay structures that do not require routing algorithms?

NICE: Hierarchical Multicast

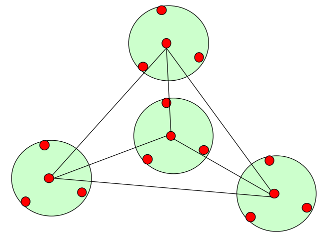
- Another scheme for overlay multicast
- Primary motivation:
  - Scalability → avoid routing overheads of Narada
  - Make low-bandwidth applications efficient
  - Large receiver-set applications like news and sports ticker
- Method: Tree based, use hierarchy
  - Group nodes into clusters
    - Nodes inside a cluster have all-to-all connectivity

NICE Clusters



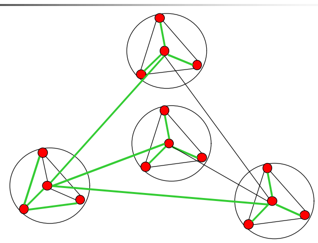
- Nodes are added to the cluster which is “closest”
- Geographic center of cluster is chosen as a leader
- Cluster membership varies between k and 3k-1
- Cluster is split if number exceeds 3k – 1

NICE Hierarchy



- Group cluster leaders into higher-level cluster, elect its leader and process continues to form a hierarchy
- When a node joins a cluster, exceeds the cluster limit, and forms two clusters ➔ new leaders added to the higher level cluster

NICE Multicast



- Any node in the system can initiate a multicast
- Multicast data would travel up and down the tree links to all of the cluster leaders
- Cluster leaders propagate data to all of its cluster members

NICE Protocol Operations

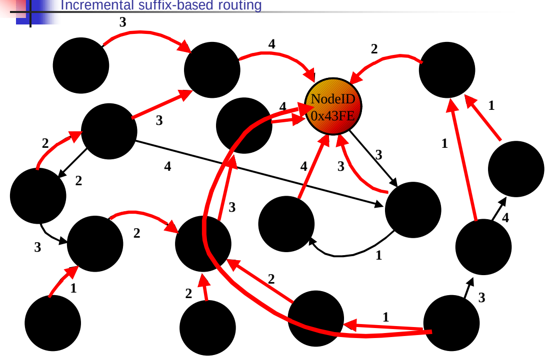
- Member join
  - Find tree leader
  - Walk down the tree, at each step find a cluster member which is closest to the joining node
  - Eventually join the lowest level cluster
  - Might be elected as a leader ➔ might have to replace existing leader in higher level cluster
- Cluster split: if joining node exceeds cluster size (3k-1)
- Member departure
  - Cluster size might drop below threshold ➔ perform cluster merge with some other cluster
- Cluster merge
  - Balance the number of nodes between two clusters
- Cluster refine
  - Occasionally nodes seek to find better clusters

Overlay Multicast using DHTs

- Nodes in DHTs form multiple implicit tree structures
- Each node in a DHT has limited degree
- DHT maintenance costs are low:
  - Typically O(log n) insertion and deletion costs
  - Typically, there are no bottlenecks in the system

Tapestry Mesh

Incremental suffix-based routing



Tapestry-based Multicast (Bayeux)

- Assume tapestry network routing tables are optimized:
  - If multiple nodes can be used to fill in a routing table entry, use the "best" node
  - Usually accomplished with limited information
- Create a "key" for each multicast
- Map the "key" to the node that hashes close to the "key"
- This node will serve as the multicast root
- Send data to this node
- Each node interested in the multicast:
  - Routes a request to the root (similar to locating a key value)
  - Requests from different nodes might intersect at different points in the system
  - Nodes at intersection points remember who "subscribes" to a certain piece of data

Announcements

- Final exam on Dec. 16th
  - Time: 2:00 – 5:00
  - Location: DL 220
  - Open book, open notes exam

Course Wrapup

Theoretical Topics

Basic distributed algorithms
(message/time complexity,
liveness/safety issues)
Asynchronous algorithms (GHS-MST)
Reasoning about distributed systems
(clocks, consistent cuts,
snapshots, global predicates)
Consensus (synchronous/asynchronous,
fail-stop/byzantine,
impossibility results, Paxos)

Distributed systems in practice

P2P file-sharing systems
Distributed hash tables
Routing algorithms in Internet
Ad-hoc routing algorithms
Security attacks and securing
distributed computations
Overlay networks

Acknowledgements:
- Course material: Attiya-Welch, Lynch, Coulouris-Dollimore-Kindberg
- Lecture notes derived from graduate courses taught by:
  Jennifer Welch (Texas A&M), Srini Seshan (CMU), Lorenzo Alvisi (UT-Austin), Hari Balakrishnan (MIT)

Final Thoughts

- Field has matured substantially in the last few years:
  - Previously, there were two communities that didn't quite interact with each other:
    - Theoretical community designed complex algorithms
    - Operating systems community designed distributed file systems and cluster operating systems
  - Recently, focus has been on designing algorithms that were immediately put into practice
  - Highly sophisticated execution:
    - Most systems are analyzed, simulated, and implemented!
- Challenges:
  - Issues of scale
  - More complex services (not just object location and multicast!)
  - Autonomous entities: not just fail-stop/byzantine node, but selfish agents
  - Securing distributed systems