Peer-to-Peer Systems

Reducing the load

Michael Welzl michael.welzl@uibk.ac.at

DPS NSG Team http://dps.uibk.ac.at/nsg
Institute of Computer Science
University of Innsbruck, Austria

Reducing the load

1. Storage in nodes
   - can be addressed with Load Balancing
2. Network communication
   - can be addressed with Overlay Multicast
   - or multicast in general, but we don't usually have it at the IP level

- Note, for both cases:
  reducing load increases scalability of P2P systems

Load Balancing

- Standard assumption of DHTs: uniform key distribution
  - Hash function
  - Every node with equal load
  - No load balancing is needed
- Equal distribution: system should be optimally balanced
  - Nodes across address space
  - Data across nodes
    - Load of each node should be around 1/N of the total load
    - Else the node is overloaded (heavy) or light
- But is this assumption justifiable?
  - Analysis of distribution of data using simulation

Chord analysis

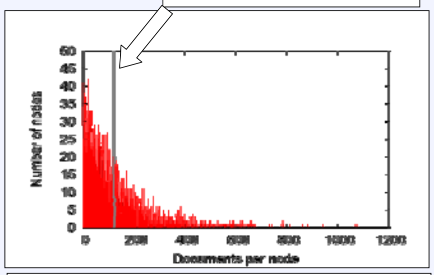
- Example
  - 4,096 nodes, 500,000 documents
  - Optimum: 122 documents per node

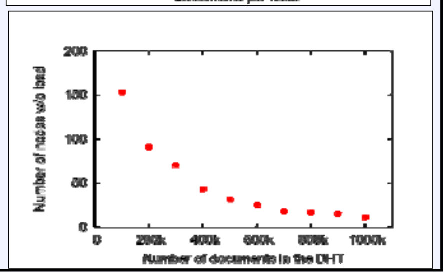
⇒ No optimal distribution in Chord w/o load balancing

- Number of nodes without storing any document
  - 4,096 nodes
  - 100,000 to 1,000,000 documents

⇒ Significant number of nodes without any load

Optimal distribution of documents across nodes



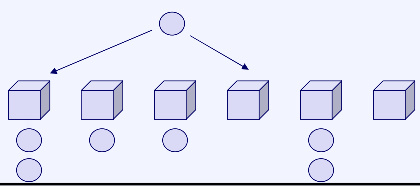


Some algorithms

- Power of Two Choices (Byers et. al, 2003)
- Virtual Servers (Rao et. al, 2003)
- Thermal-Dissipation-based Approach (Rieche et. al, 2004)
- A Simple Address-Space and Item Balancing (Karger et. al, 2004)

Power of two choices

- Basic concept:
  - One hash function for all nodes (h0); multiple for data (h1, h2, h3, …hd)
  - Two variants: with or without pointers
- Inserting Data
  - Results of all hash functions are calculated (h1(x), h2(x), h3(x), …hd(x))
  - Data is stored on the retrieved node with the lowest load
    - Alternative: other nodes stores pointer
  - Owner of data must periodically insert the document
    - Prevent removal of data after a timeout (soft state)

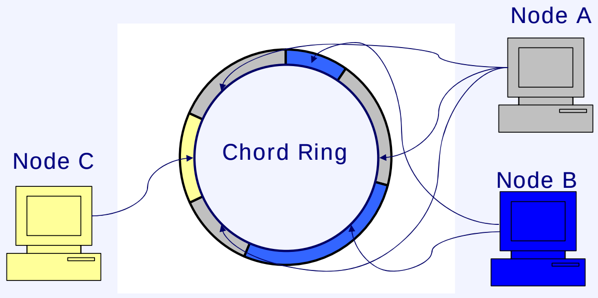


Power of two choices /2

- Retrieving
  - Without pointers
    - All hash functions are calculated; request all possible nodes in parallel
    - One node will answer
  - With pointers
    - Request only one of the possible nodes
    - Node can forward the request directly to the final node
- Main algorithm advantage: simplicity
- Disadvantages
  - Message overhead when inserting data
  - With pointers
    - Additional administration of pointers; more load
  - Without pointers
    - Message overhead at every search

Virtual Server

- Each node responsible for several intervals (“virtual server“)

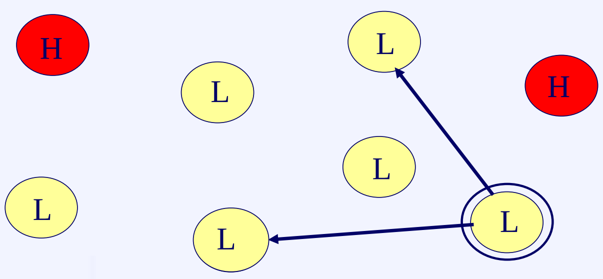


Transferring a Virtual Server

- Rules for transferring a virtual server (from heavy node to light node)
  1. The transfer of a virtual server does not make the receiving node heavy
  2. Virtual server is the lightest virtual server that makes the heavy node light
  3. If there is no virtual server whose transfer can make a node light, the heaviest virtual server from this node would be transferred
- Due to these rules, there are log (n) virtual servers
- Different possibilities to change servers:
  - one-to-one, one-to-many, many-to-many
- Copy of an interval is like removing and inserting a node in a DHT

Scheme 1: one-to-one

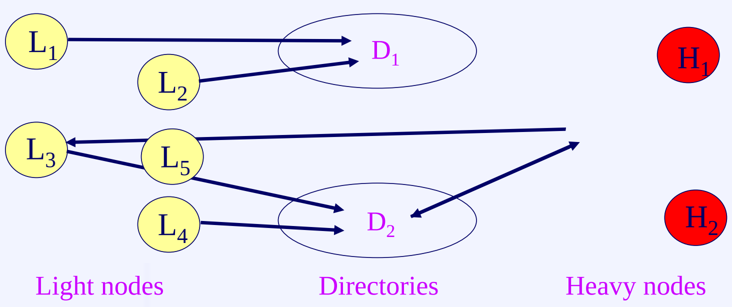
- Light node picks a random ID; contacts the node x responsible for it
- Accepts load if x is heavy



[Rao 2003]

Scheme 2: one-to-many

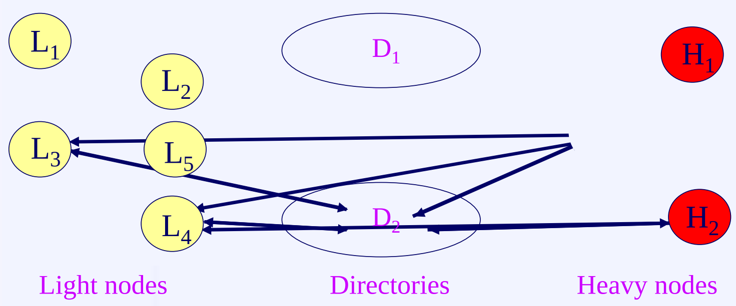
- Light nodes report their load information to directories
- Heavy node H gets this information by contacting a directory
- H contacts the light node which can accept the excess load



[Rao 2003]

Scheme 3: many-to-many

- Many heavy and light nodes rendezvous at each step
- Directories periodically compute the transfer schedule and report it back to the nodes, which then do the actual transfer



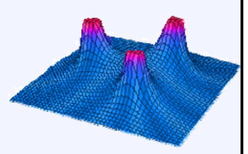
[Rao 2003]

Virtual Server: conclusion

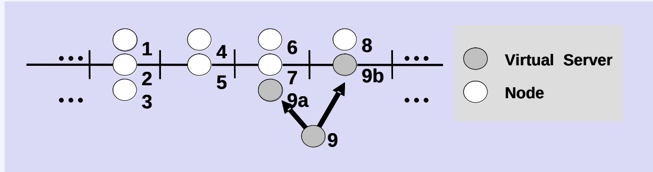
- Advantages
  - Easy shifting of load
    - Whole Virtual Servers are shifted
- Disadvantages
  - Increased administrative and messages overhead
    - Maintenance of all Finger-Tables
  - Much load is shifted

Thermal-Dissipation-based Approach

- Moving content among peers: similar to the process of heat expansion
  - Several nodes in one interval ⇒ DHT more fault tolerant
- Fixed positive number f indicates how many nodes have to act within one interval at least

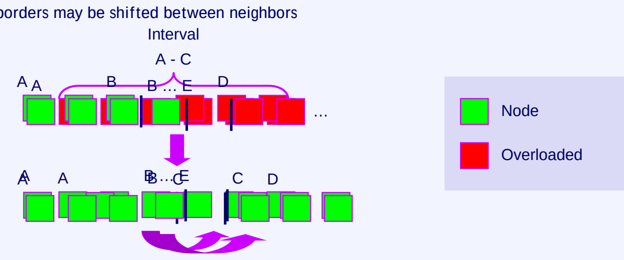


- Procedure
  - First node takes random position
  - A new node is assigned to any existing node
  - Node is announced to all other nodes in same interval
  - Copy of documents of interval (more fault tolerant system)



Algorithm

- Nodes can balance the load with other intervals
  - Three methods
    1. 2f different nodes in same interval and nodes are overloaded
       - Interval is divided
    2. More than f but less than 2f nodes
       - Release some nodes to other intervals
    3. Interval borders may be shifted between neighbors



Address-space balancing

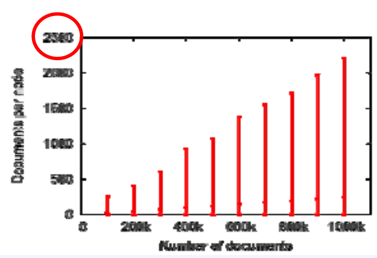
- Each node has a fixed set of O(logn) possible positions
  - "virtual nodes"
- Chooses exactly one of those virtual nodes
  - this position becomes active
  - This is the only position where it actually operates
- Node's set of virtual nodes depends only on the node itself
  - Computed as hashes: h(id,1),h(id,2), . . . ,h(id,logn)
- Each (possibly inactive) virtual node "spans" a certain range of addresses
  - Between itself and its succeeding active virtual node
- Each real node has activated the virtual node
  - Which spans the minimal possible address space

Comparison of Load-Balancing methods

- Simulations
  - Scenario: 4,096 nodes (comparison with other measurements), 100,000 to 1,000,000 documents
  - Chord; m= 22 bits
  - Consequently, 2^{22} = 4,194,304 nodes and documents
- Hash function
  - sha-1 (mod 2^m)
  - random
- Analysis
  - Up to 25 runs per test

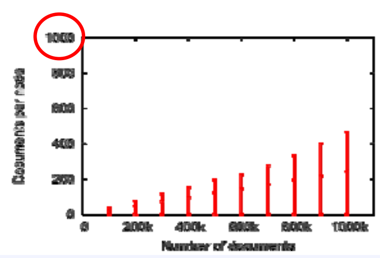
Results

- Without load balancing



+ Simple
+ Original
– Bad load balancing

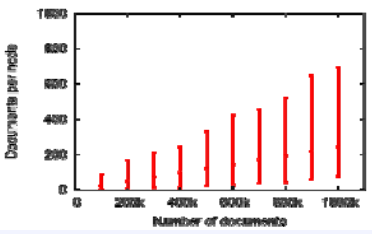
- Power of Two Choices



+ Simple
+ Lower load
– Nodes w/o load

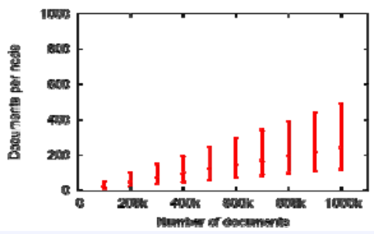
Results /2

- Virtual server



+ No nodes w/o load
- Higher max. load than Power of Two Choices

- Thermal-Dissipation



+ No nodes w/o load
+ Best load balancing
- More effort (but redund.)

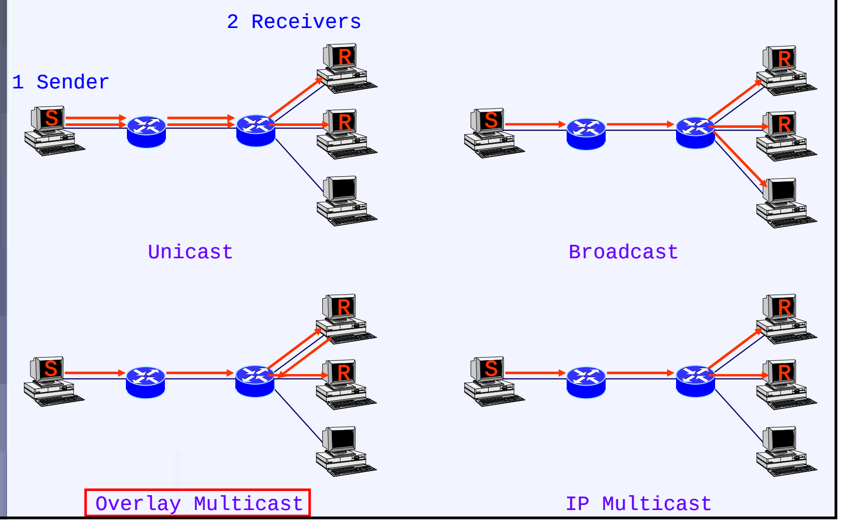
Other methods

- Fragmentation
  - Idea: Divide each object into chunks, store chunks individually
  - One chunk is much smaller than a file, hence load is balanced better, since chunks are stored on different peers
  - Achieves overall load balancing (goal 1 from above)
  - Network coding can be applied here
- Overflow
  - Idea: Allow peers to refuse requests
  - Request passed on to the next winner (eventually to outside)
  - Allows a peer to decide how much traffic to handle
  - Achieves goal number 2 from above
- Fragmentation + Overflow
  - Use both approaches

Load Balancing summary

- Need of Load Balancing for DHTs
  - Otherwise, load is severely unbalanced
- Several fundamental techniques to ensure balanced load
  - Power of Two Choices (Byers et. al, 2003)
  - Virtual Servers (Rao et. al, 2003)
  - Thermal-Dissipation-based Approach (Rieche et. al, 2004)
  - A Simple Address-Space and Item Balancing (Karger et. al, 2004)
- Alternatives: fragmentation, overflow approaches
  - Fragmentation (based on redundancy) appears to work well
  - Pure overflow approach allows individual peers to reduce their load at a cost of increased load to others
  - Combination of overflow with fragmentation possible, works very well

Reducing network load: overlay multicast

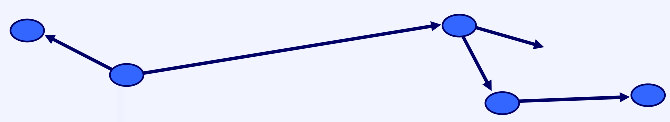


Key concerns with IP multicast

- Scalability with number of groups
  - Routers maintain per-group state
    - Analogous to per-flow state for QoS guarantees
  - Aggregation of multicast addresses is complicated
- Supporting higher level functionality is difficult
  - IP Multicast: best-effort multi-point delivery service
  - End systems responsible for handling higher level functionality
  - Reliability and congestion control for IP Multicast complicated
- Inter-domain routing is hard
- Deployment is difficult and slow
  - ISP's reluctant to turn on IP Multicast

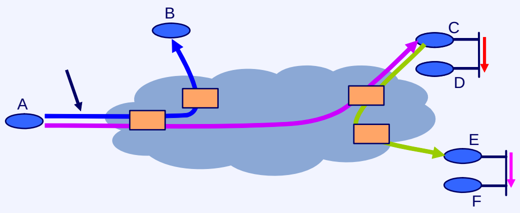
Potential benefits of Overlay Multicast

- Scalability (number of sessions in the network)
  - Routers do not maintain per-group state
  - End systems do, but they participate in very few groups
- Easier to deploy
- Potentially simplifies support for higher level functionality
  - Leverage computation and storage of end systems
  - For example, for buffering packets, transcoding, ACK aggregation
  - Leverage solutions for unicast congestion control and reliability

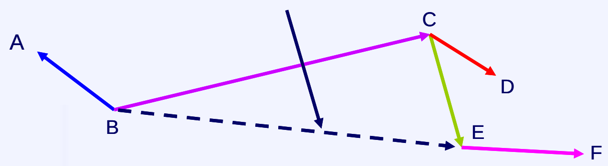


Performance concerns - metrics

- Duplicate Packets on physical links:
  - Bandwidth Wastage

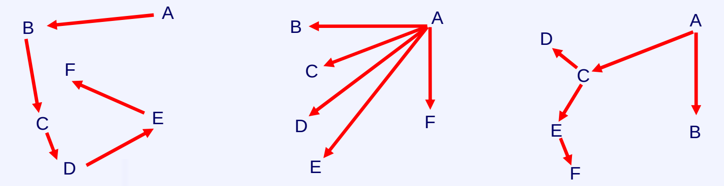


  - Stress: # times a packet traverses a physical link
- Increase of End-to-End-Latency
  - Direct links may be shorter



What is an efficient overlay tree?

- The delay between the source and receivers is small
- Ideally, the number of redundant packets on any physical link is low
- Typical heuristic:
  - Every member in the tree has a small degree
  - Degree chosen to reflect bandwidth of connection to Internet



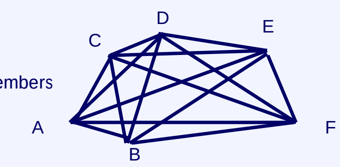
High latency | High degree (unicast) | "Efficient" overlay

Multicast: issues and approaches

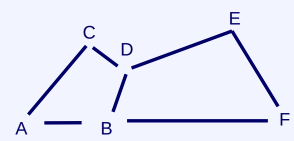
- Self organization is hard
  - Dynamic changes in group membership
    - Members join and leave dynamically
    - Members may die
  - Limited knowledge of network conditions
    - Members do not know delay to each other when they join
    - Overlay must self-improve as more information available
  - Dynamic changes in network conditions
    - Delay between members may vary over time due to congestion
- Overlay multicast approaches
  - Unstructured approaches
    - Narada: "A case for end-system multicast"
    - NICE: more scalable using clustering (not considered here)
  - Approaches based on DHTs
    - Pastry-based: Scribe
    - CAN-Multicast
    - Internet Indirection Infrastructure (i3)
    - … and others based on Chord, Tapestry etc.

Narada Design

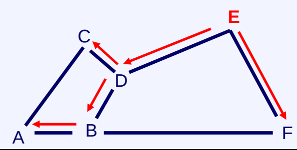
- Step 0
  - Maintain a complete overlay graph of all group members
  - Links correspond to unicast paths
  - Link costs maintained by polling



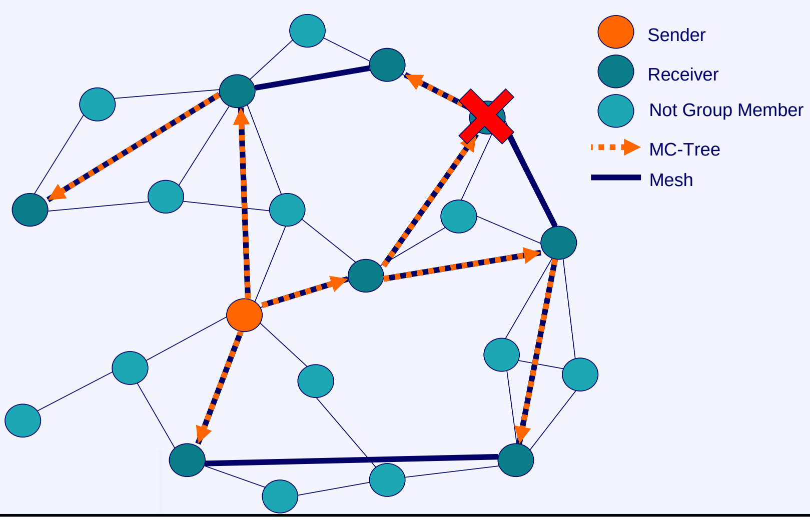
- Step 1
  - "Mesh": Subset of complete graph (may have cycles and includes all group members)
  - Members have low degrees
  - Shortest path delay between any pair of members along mesh is small



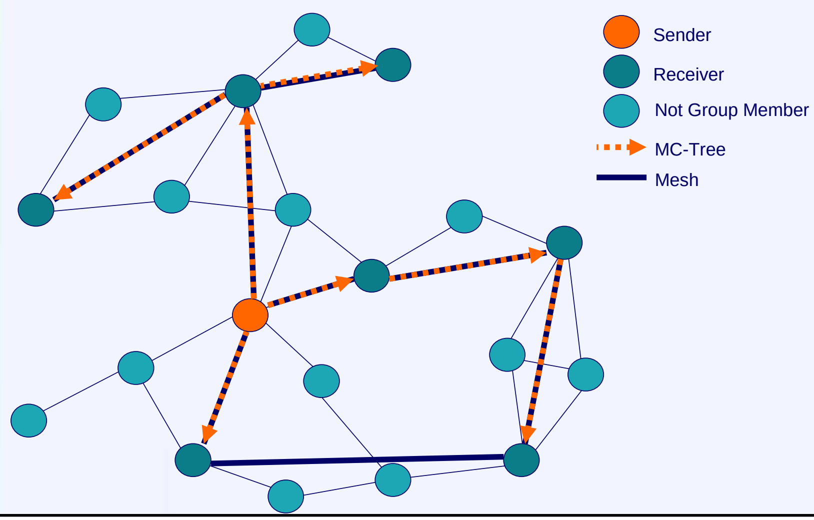
- Step 2
  - Build spanning tree within the mesh
  - Constructed using well known routing algorithms
  - Members have low degrees
  - Small delay from source to receivers



Narada Example



Narada Example

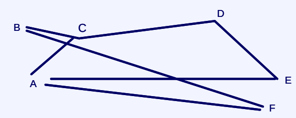


Narada Components

- 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

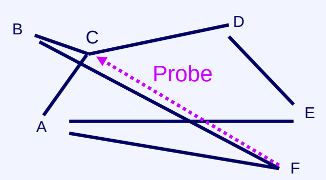
- Members periodically probe other members at random
- New Link added if utility gain of adding link > Add Threshold
  - Based on: number of members to which routing delay improves, how significant the improvement in delay to each member is
- Members periodically monitor existing links
- Existing Link dropped if cost of dropping link < Drop Threshold
  - Based on number of members to which routing delay increases, per neighbor
- Add/Drop Thresholds are functions of:
  - Member's estimation of group size
  - Current and maximum degree of member in the mesh



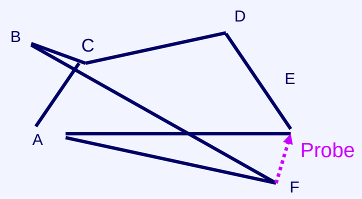
A poor overlay topology

Desirable properties of heuristics

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

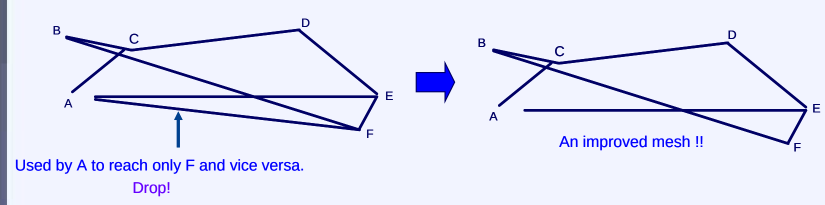


Delay improves to C, D
but marginally.
Do not add link!



Delay improves to D, E
and significantly.
Add link!

Adding / dropping links contd., overhead



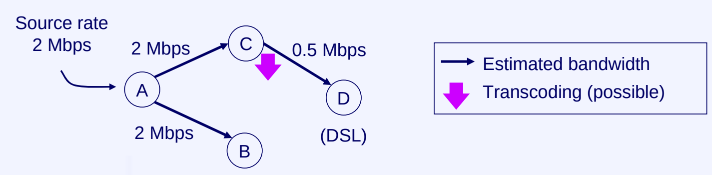
- Two sources of overhead
  - pairwise exchange of routing and control information
  - polling for mesh maintenance
- Claim: ratio of non-data to data traffic grows linearly with group size.
- Narada is targeted at small groups

End System Multicast (ESM)

- Goal: enable conferencing on the Internet based on Narada
  - Study in context of real-world applications
  - Performance acceptable even in a dynamic and heterogeneous environment
- ESM = first detailed Internet evaluation to show the feasibility of ESM
- Why conferencing?
  - Important and well-studied
    - Early goal and use of multicast (vic, vat)
  - Stringent performance requirements
    - High bandwidth, low latency
  - Representative of interactive applications
    - E.g., distance learning, on-line games

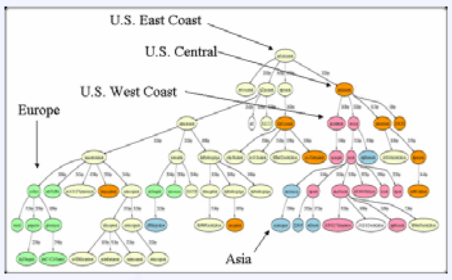
Supporting conferencing in ESM

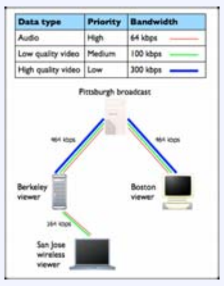
- Framework
  - Bandwidth estimation
  - Adapt data rate to bandwidth est. by packet dropping
- Objective
  - High bandwidth and low latency to all receivers along the overlay



Enhancements of Overlay Design

- Two new issues addressed
  - Dynamically adapt to changes in network conditions
  - Optimize overlays for multiple metrics
    - Latency and bandwidth
- Narada is used in ESM
  - ESM has been successfully used for streaming conference talks
  - Available from http://esm.cs.cmu.edu/





Scribe

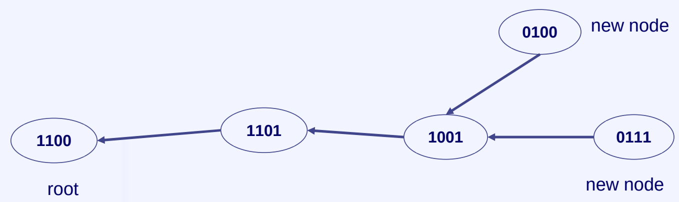
- Scalable application-level multicast infrastructure built on top of Pastry
- API
  - create (credentials, group-id)
    - create a group with the group-id
  - join (credentials, group-id, message-handler)
    - join a group with group-id.
    - Published messages for the group are passed to the message handler
  - leave (credentials, group-id)
    - leave a group with group-id
  - multicast (credentials, group-id, message)
    - publish the message within the group with group-id
  - credentials are used throughout for access control

Scribe System

- A Scribe node may create a group, join a group, be the root of a multicast tree or act as a multicast source
- Scribe messages: CREATE, JOIN, LEAVE, MULTICAST (publish a message to the group)
- Multicast groups: a Scribe group per session
  - Unique group-id; multicast tree to disseminate messages
  - Root of tree = rendezvous point
  - GroupID = hash of group's textual name concatenated with it's creator's name
- Create Group
  - Send a CREATE message with the group-id as the key
  - Pastry delivers message to root (key)
  - This node becomes the rendezvous point
  - deliver method checks and stores credentials and also updates the list of groups

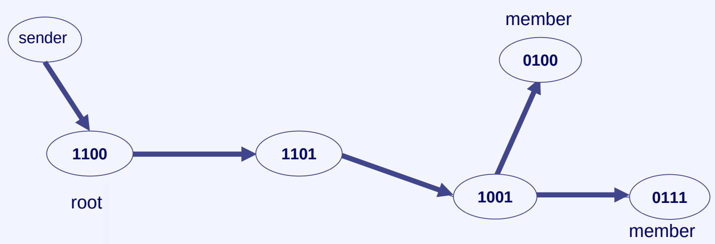
Join Group

- Send JOIN message with group-id as key
- Pastry routes to rendezvous point
  - If intermediate node is forwarder
    - Add the node as its child
  - If intermediate node is not a forwarder
    - Creates child table for the group, and adds the node
    - Sends a JOIN towards the rendezvous point.
  - Terminates JOIN message from the child



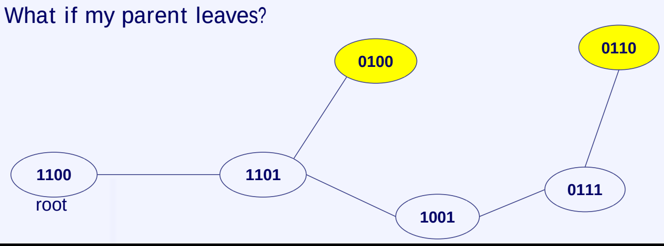
Multicast Message

- Multicast a message to the group
  - Scribe node sends MULTICAST message to the rendezvous point
  - A node caches the IP address of the rendezvous point so that it does not need Pastry for subsequent messages
  - Single multicast tree for each group
  - Access control for a message is performed at the rendezvous point



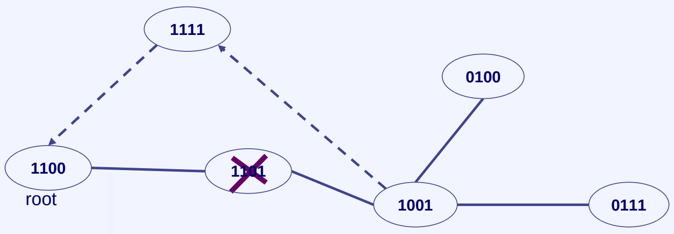
Leave Group

- Scribe node records locally that it left the group
- If the node has no children in its table, it sends a LEAVE message to its parent
  - The message travels recursively up the multicast tree
  - The message stops at a node which has children after removing the departing node
- What if my parent leaves?



Multicast Tree Repair

- Broken link detection and repair
  - Non-leaf nodes send heartbeat message to children
  - Multicast messages serve as implicit heartbeat
  - If child does not receive heartbeat message
    - assumes that the parent has failed
    - finds a new route by sending a JOIN message to the group-id, thus finding a new parent and repairing the multicast tree

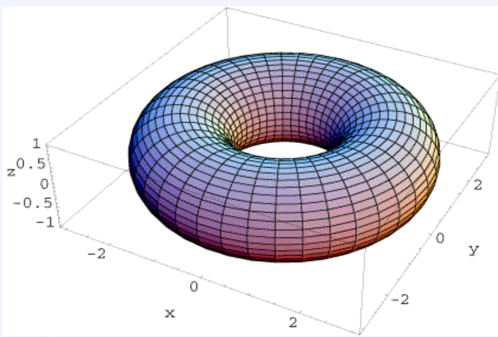


Multicast Tree Repair, Bottleneck Remover

- Rendezvous point failure
  - The state associated with a rendezvous point is replicated across k closest nodes
  - When the root fails, the children detect the failure and send a JOIN message which gets routed to a new node-id numerically closest to the group-id
- Fault detection and recovery is local and accomplished by sending minimal messages
- Bottleneck remover
  - All nodes may not have equal capacity in terms of comput. power and bandwidth
  - Under high load conditions, the lower capacity nodes become bottlenecks
- Solution: Offload children to other nodes
  - Choose the group that uses the most resources
  - Choose a child of this group that is farthest away
  - Ask the child to join its sibling which is closest in terms of delay
- This improves performance, but increases link stress for joining

CAN-Multicast

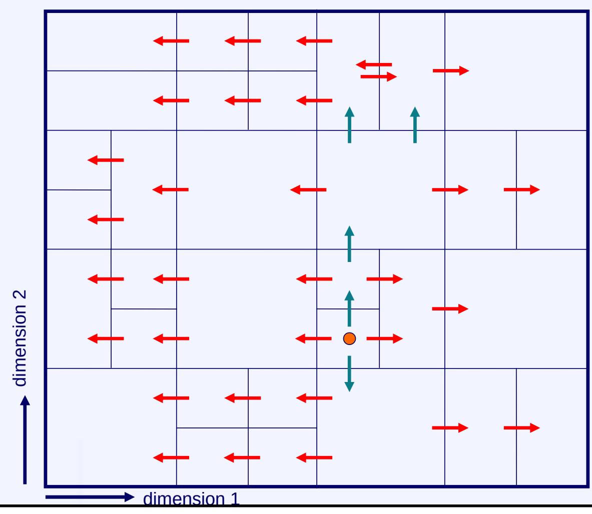
- Content-Addressable Networks
  - Benefit from the geometry of the CAN-overlay



CAN Multicast Routing

- Multicast Forwarding in CAN DHTs
  - Source sends message to all neighbors
  - If a neighbor receives the message along dimension i, it forwards the message to …
    - …all neighbors along the i[th] dimension (to the opposite direction the message was received – just simple forwarding along dimension i)
    - …all neighbors, whose zones are neighboring in dimension 1…(i-1)
  - If a packet traversed half of the address space along dimension i, stop forwarding along dimension i
- Advantage:
  - If space is well (equally) partitioned, packets are not received multiple times

CAN Multicast Routing



Overlay multicast summary

- End-system-based Multicast
  - No support by infrastructure required
  - Shift of complexity to end-systems
  - Individual metrics and individual adaptations (e.g. transcoding) possible
  - Higher link usage and end-to-end latency (compared to layer-3 approach)
- Two different design choices
  - Unstructured Multicast Overlay (Mesh-first)
    - Does not scale well
    - Good adaptation to network topology
  - Structured Multicast Overlay
    - Scalable
    - Adaptation to network topology is harder

References / acknowledgments

- Slides from:
  - Jussi Kangasharju
  - Klaus Wehrle