Guaranteed Broadcasting Using SPON: Supervised P2P Overlay Network

Chris Riley, Christian Scheideler
Computer Science Department
Johns Hopkins University
{chrisr, scheideler}@cs.jhu.edu

***Abstract*— One important application for dynamic networks is broadcast or multicast communication, where a single message is sent to all members of a dynamic set of recipients. This functionality is required by many data-sharing protocols, as well as applications including media streaming, online gaming, teleconferencing, and distance education. This paper proposes a tree-based network called SPON which manages group updates and supports efficient broadcasting. SPON uses a supervisor peer to maintain the network during node arrivals and departures and routes broadcasts using direct connections between nodes.**

I. Introduction

Broadcast communication is an essential component of many networking applications, including audio/video streaming, multiplayer online gaming, distance education, teleconferencing, and many data lookup protocols. The group of recipients of the broadcast often changes over time, and the challenge of the algorithm designer is to provide a means of broadcasting that tolerates these changes while keeping node responsibilities and communication costs low. When a central server can store a list of all group members and send the broadcast directly to each one, the problem is trivial. But this greatly limits scalability and requires a very expensive server. It is better to organize a broadcast group as an *overlay network*, an application-layer graph in which each vertex is a group member and each edge represents a path through a lower layer network such as the Internet; broadcasts in an overlay network are sent along the edges of the graph.

Two dominant models of network construction are client/server systems and peer-to-peer systems. Client/server systems use a single central server and multiple small clients who communicate with the server. In peer-to-peer systems clients communicate with each other, and traditionally there is no central node. We propose an approach that combines the advantages of these systems, called the "supervised peer" approach. Our approach uses a single reliable but low-powered supervisor node to control the organization of the other nodes in the network. We present a tree-based structure and algorithms for maintaining this structure within the supervised peer model. Our system, SPON, is used for dynamic broadcasting, and supports node join and leave operations and message broadcasts. SPON is capable of guaranteeing broadcasts under an adversarial event model (where an adversary is allowed to request arbitrary events up to some frequency limits).

A. Our Results

SPON is capable of performing reliable broadcasting in unreliable networks. We consider broadcasting to be reliable when any broadcast whose lifetime is entirely within a node's lifetime is guaranteed to be received by that node; this statement is nontrivial since we also bound the lifetime of a broadcast. When only *graceful* departures are allowed (when a node provides sufficient advance notice for departures), SPON has the following performance:

- SPON performs join and leave operations in $O(1)$ time using $O(1)$ work[1].
- SPON performs broadcast operations in $O(\log n)$ time using $O(n)$ work.

SPON also performs well if *ungraceful* departures are allowed, even against operations selected by an adversary, while keeping the same broadcast guarantee. Most systems measure their performance only against random node failures, but SPON's reliable supervisor allows it to achieve much greater fault tolerance. Using the model of adversarial queueing theory (*e.g.* [1], [3]), against a (λ, T)-bounded adversary SPON has the following performance:

- SPON is *stable* for $\lambda \leq c$ for some constant c.
- SPON performs join and leave operations in $O(\lambda T + \log n)$ time using $O(1)$ amortized work.
- SPON performs broadcast operations in $O(\lambda T + \log n)$ time using $O(n)$ work.

Furthermore, each node uses $O(\log n)$ storage and has degree $O(1)$ except for the supervisor who uses $O(\lambda T + \log n)$ storage and has degree $O(\log n)$.

B. Related Work

Peer-to-peer networks have been studied extensively for information-sharing purposes. The first popular strategy, Napster[14], used a centralized index, which requires a server node with $O(n)$ storage (where n is the number of nodes in the network). Flooding-based, completely decentralized approaches were introduced in an effort to avoid censorship and regulation and to distribute load ([4], [7]); most of these reduce local storage to a constant or $O(\log n)$ for every node, but have no guarantees on the time or work required or even that a message will reach a substantial fraction of nodes (as

[1]For a definition of work, see Section II-A

a normal time-to-live is set smaller than the possible $O(n)$ network diameter to avoid exponential work).

Much current peer-to-peer research studies the data lookup problem, using distributed hashing techniques for data storage and retrieval in symmetric decentralized networks. This group includes Chord[21], Pastry[20], Tapestry[9], CAN[18], Kademlia[13], and Viceroy[12]. These strategies use structured placement and sometimes replication for data items to ease later location. This can reduce time and work to $O(\log n)$ for search operations, which improves on any flooding strategy for searching where $\Omega(n)$ work is required since in the worst case all nodes must be contacted.

A randomized, supervised network was presented by Pandurangan, Raghavan, and Upfal [15] which maintains a logarithmic diameter and a constant degree for all nodes (and thus $O(1)$ storage, $O(\log n)$ time, and $O(n)$ work), but can only give probabilistic guarantees of connectivity. Their work also did not consider information sharing problems such as reliable broadcasting.

Recently researchers have begun to consider the specific problem of managing media streaming over peer-to-peer networks, also known as application-layer multicasting. The SpreadIt architecture of Deshpande *et al* [5] uses a tree-based broadcast mechanism, based on a greedy assignment of new nodes to unsaturated servers; the paper does not upper bound the time or work to find an unsaturated server (a process necessary in all node insertions and deletions). NICE [2], a hierarchical clustering approach based on a tree of clusters of peers, requires logarithmic time and work for node insertion, and may require extensive correction time for node failure (when a cluster leader departs ungracefully and a new leader must be determined). The hierarchical clustering approach of Tran *et al* [22], based on NICE, bounds failure correction time, but still requires logarithmic time and work to insert a node, and does not discuss network reconnection after the parallel failure of arbitrary sets of nodes. El-Ansary *et al* [6] study efficient broadcasting in a structured DHT-based network, but they consider logarithmic DHT's such as Chord and therefore require at least logarithmic work to insert a node.

The Overcast system [10] uses a supervised tree-building protocol to build a system for single-source multicasting; they do not analyze their tree building protocol for its theoretical performance. The Bullet system [11] uses a mesh topology and divides the message into blocks which are sent along different edges of the mesh with redundancy; this model may not be usable in some broadcast applications, particularly realtime applications. The RMTP protocol [16] builds a tree structure of local clusters of nodes in order to achieve reliability through selective acknowledgements and retransmissions; their work focuses mostly on reliability within the cluster rather than on efficient cluster organization to handle highly dynamic environments. On top of reliable unordered multicast systems using multiple sources the *distributed swap* coordination problem can be used to force ordering [8]; our system bypasses this issue by forcing broadcasts to begin with the supervisor, which forces ordered transmission.

II. The SPON tree

A. Requirements

We present a supervised peer-to-peer overlay network for efficient dynamic broadcasting. To support such a system we use the following operations:

- s.JOIN(p): supervisor s inserts peer p into the system
- s.LEAVE(p): supervisor s removes peer p from the system
- s.BROADCAST(m): supervisor s wishes to send m to all nodes

We assume that nodes contact the supervisor to request their insertion and removal. In practice, any node can be contacted to let a node join the system, and it will forward the request to the supervisor; similarly, any node can request a broadcast transmission by first sending the message to the supervisor. This formulation is for graceful departures; we will omit ungraceful departure details for space.

The network layout must maintain certain desirable topological properties, particularly:

- **Degree:** The degree of each node should be kept low to reduce update costs for join and leave operations and to increase scalability.
- **Diameter:** The diameter of the network should be small to enable fast broadcasting.

Furthermore, we would like the operations to be efficient. One way to evaluate efficiency is to consider the *message graph* for each operation. The operation message graph $H = (M, E)$ contains a message $m \in M$ for each message sent during the course of the operation and a directed edge $(m_1, m_2) \in E$ when message m_1 causes message m_2 to be sent. For example, a message sent to a node for a broadcast causes messages to be sent along other links. Within this framework we can define important cost measures including:

- **Work:** This measures the total number of messages for each operation, or $|M|$. Low work leads to faster operations and lower network load.
- **Time:** This measures the total number of time steps required to complete an operation; it is generally proportional to the length of the longest directed path in H. Low time for an operation allows greater network scalabilty and throughput.

B. Topology

The system topology forms a tree. The supervisor maintains $2 \log n$ root slots divided into $\log n$ pairs labeled from 0 to $\log n - 1$; each slot can contain a connection to a node or can be empty. Nodes in these slots are called *root nodes*, and adjacent root nodes are connected. The network is organized according to the following two rules:

Invariant 2.1: A peer stored in a slot of pair i is the root of a complete binary tree of peers of depth i.

Invariant 2.2: At most one slot pair is fully occupied, and below this pair there is no occupied slot.

Theorem 2.3: Except for the supervisor node degree is at most 4; the supervisor has degree $O(\log n)$. The network diameter is $O(\log n)$.

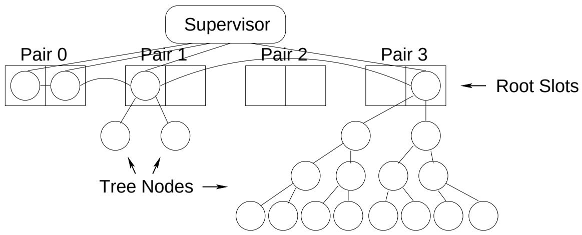

Fig. 1. A sample network

Proof: Network diameter is $O(\log n)$ since the maximum distance between any two nodes is from the bottom of one tree across the root nodes down to the bottom of another tree; each of these distances is $O(\log n)$. ■

C. Join and leave operations

We assume that invariants 2.1 and 2.2 hold. To insert a node v into the system, the supervisor performs one of the following:

- If there is a fully occupied slot pair j, insert v into an open slot in pair $j+1$ and move the nodes from pair j to be children of v.
- Otherwise, insert v into an open slot in pair 0.

The supervisor can process a node join using only $O(1)$ messages. Clearly the operations preserve invariants 2.1 and 2.2.

To remove a node from the system, the supervisor removes the lowest root node and uses it to replace the departing node (if they are different). To remove the lowest root node w, the supervisor performs one of the following:

- If w is in pair 0, just remove it.
- Otherwise, if w is in pair $i > 0$, move w's children to the open root slots in pair $i-1$, and remove w from the system.

These also require only $O(1)$ messages. Clearly invariant 2.1 is preserved. It also holds that:

Lemma 2.4: Invariant 2.2 holds after each remove operation.

Proof: Consider the first remove operation that produces a violation by making two full slot pairs; before the operation the invariant holds. Then before this operation, a root node above the lower of the slot pairs was removed, which implies that it was below the other full slot pair, which is a contradiction. Suppose instead that there is a remove operation that produces a node below a full slot pair. If the full slot pair was produced by the remove operation by removing a node, then the lowest node was not removed, which is a contradiction. If the full slot pair was not produced by the remove operation, then there was some lower node below the slot pair which was removed, which implies that there was a node below the slot pair before the operation, which is a contradiction. ■

The cost of the operations is clear from the algorithms:

Theorem 2.5: On a network of n nodes, JOIN and LEAVE require $O(1)$ work and $O(1)$ time.

III. BROADCAST OPERATIONS

A. Graceful departures only

We detail the operation of broadcasting under the assumption that node departures are graceful. Broadcasts are sent by the supervisor first to the lowest root node, and from there through the edges in the data structure. Because they are sent through a tree from a single point, they arrive in an ordered fashion. We suppose that a leave operation is requested by a node v wishing to depart; since departures are graceful we do not let v depart until instructed. The following describes the new node removal process:

1) Pause broadcasts being sent to the lowest root node x and let it send all its broadcasts to its neighbors.
2) Remove x, and restart broadcast transmission to the new root node.
3) Insert x in v's place in the data structure so any broadcasts from further up the tree go to x.
4) Send any broadcasts currently held by v onto its neighbors.
5) Tell v it can depart.

We also offer the following modified join algorithm for the case when the new node is inserted as a root node not in pair 0 (and is given two child trees):

1) Move the child trees out of the root slot and insert the new node so that new broadcasts go to the new node.
2) Connect the child trees to the new node so that it can send broadcasts to them.
3) If the left child tree root was in the act of forwarding a broadcast to the right child tree root let it do so before disconnecting them.

If the new node is inserted as a root node in pair 0 the process is unchanged. Using these procedures it is easy to see:

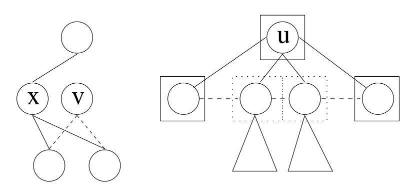

Fig. 2. Inserting x to replace v; inserting u as a root node.

Theorem 3.1: BROADCAST requires time $O(\log n)$ and work $O(n)$ where n is the number of nodes when the broadcast is released into the system.

Proof: The longest path for a broadcast is across the root nodes and down a tree, which is $3 \log n$. The only situations when a broadcast is sent to a node more than once are when the node is inserted as a replacement, which can be charged as sending the request to the departed node, so that if there are n nodes in the system when the broadcast is started it will be sent across only n edges. Nodes that join after the broadcast is started do not receive it, because they are inserted at the top of the broadcast tree where the broadcast has already passed. ■

Theorem 3.2: Broadcast requests are guaranteed to reach all nodes alive during the entire lifetime of the broadcast.

Proof: Clearly as soon as a node joins the system it begins receiving broadcasts, since each node is inserted as the lowest root node where broadcasts originate. If a node departs before some broadcast reaches it, then the node was not alive during the broadcast's entire lifespan and our guarantee does not cover it. No node can be affected by the departure of another node since the leave procedure does not drop any broadcasts. ■

B. Ungraceful departures

With modified operations, SPON can support ungraceful departures. We will outline this section; full details can be found in the technical report[19]. For the purpose of this section, we assume that all nodes, including the supervisor, have only constant bandwidth. We assume that join, leave, and broadcast operations are requested by a (λ, T)-bounded adversary, or that in any window of T time steps the adversary can request at most λT operations (though these can all be requested at the same time). We also assign sequence numbers to the broadcasts (which are still transmitted in order).

1) Node responsibilities: We require the following additional responsibilities of the nodes:

- Keep a small buffer of recent broadcasts.
- Periodically check neighbors to see if they are missing.
- Store height (which never changes except when a node is inserted as a new node or as a replacement).
- Store the sequence number of the most recent broadcast.

2) Missing nodes: If some node detects a missing neighbor, it issues a *missing node request* to the supervisor, which contains its identification, its height, and whether the missing node is a parent, child, or sibling. The supervisor collects a set of these, and processes them to determine a set of *phantom nodes* which correspond to all missing nodes in the structure. The supervisor then extracts enough nodes to take the place of the phantom nodes, discarding any extracted phantom nodes.

3) Ensuring reliable broadcasts: Many broadcasts may be waiting at a missing node to be transmitted. While at first glance it seems like the node above the missing node could store these and retransmit after the missing node is replaced, this node may fail before it can finish the transmission, and the node above it may not have kept enough. Instead, we solve the problem by ensuring that every replacement node has enough history to keep its descendants from missing any broadcasts.

In order to make this process efficient, the replacement nodes are connected into a chain, and the broadcasts are transmitted to them by the supervisor in a pipelined fashion. They retransmit any missing broadcasts to their descendants after they are inserted.

4) Performance: We have the following performance for SPON with ungraceful departures:

- SPON guarantees that each broadcast will be received by all nodes alive throughout its entire life.
- SPON is *stable*, or it can sustain constant adversarial power regardless of the size of the network.
- SPON performs join and leave operations in $O(\lambda T + \log n)$ time with $O(1)$ amortized work, and broadcast operations in $O(\lambda T + \log n)$ time with $O(n)$ work.
- The supervisor requires $O(\lambda T + \log n)$ storage; other nodes require $O(\log n)$ storage.

References

[1] M. Andrews, B. Awerbuch, A. Fernández, J. Kleinberg, T. Leighton, and Z. Liu. Universal stability results for greedy contention-resolution protocols. In *Proceedings of the 37th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 380–389, 1996.

[2] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. Technical Report UMIACS-TR 2002-53/CS-TR 4373, University of Maryland, College Park, 2002.

[3] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D. P. Williamson. Adversarial queueing theory. In *Proceedings of the 28th ACM Symposium on Theory of Computing (STOC)*, pages 376–385, 1996.

[4] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science*, 2009:46+, 2001.

[5] H. Deshpande, M. Bawa, and H. Garcia-Molina. Streaming live media over a peer-to-peer network. Technical Report 2001-31, Stanford University, 2001.

[6] S. El-Ansary, L. O. Alima, P. Brand, and S. Haridi. Efficient broadcast in structured p2p networks. In *Second International Workshop on Peer-to-Peer Systems*, 2003.

[7] Gnutella. http://gnutella.wego.com.

[8] M. Herlihy, S. Tirthapura, and R. Wattenhofer. Ordered multicast and distributed swap. *Operating Systems Review*, 35(1):85–96, 2001.

[9] K. Hildrum, J. D. Kubiatowicz, S. Rao, and B. Y. Zhao. Distributed object location in a dynamic network. In *Proceedings of the Fourteenth ACM Symposium on Parallel Algorithms and Architectures*, pages 41–52, Aug. 2002.

[10] J. Jannotti, D. Gifford, K. Johnson, F. Kaashoek, and J. O'Toole. Overcast: Reliable multicasting with an overlay network. In *4th Symposium on Operating System Design and Implementation*, 2000.

[11] D. Kosti, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: high bandwidth data dissemination using an overlay mesh. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 282–297. ACM Press, 2003.

[12] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing*, 2002.

[13] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, Mar. 2002.

[14] Napster. http://www.napster.com.

[15] G. Pandurangan, P. Raghavan, and E. Upfal. Building low-diameter p2p networks. In *IEEE Symposium on Foundations of Computer Science*, pages 492–499, 2001.

[16] S. Paul, K. K. Sabnani, J. C.-H. Lin, and S. Bhattacharyya. Reliable multicast transport protocol (RMTP). *IEEE Journal of Selected Areas in Communications*, 15(3):407–421, 1997.

[17] C. G. Plaxton and R. Rajaraman. Fast fault-tolerant concurrent access to shared objects. In *IEEE Symposium on Foundations of Computer Science*, pages 570–579, 1996.

[18] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM*, pages 161–172. ACM Press, 2001.

[19] C. Riley and C. Scheideler. Guaranteed broadcasting using SPON: Supervised peer overlay network. Technical Report, Johns Hopkins University, 2003.

[20] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM Int'l Conf. on Distributed Systems Platforms*, 2001.

[21] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.

[22] D. Tran, K. Hua, and T. Do. Zigzag: An efficient peer-to-peer scheme for media streaming. *To appear at IEEE INFOCOM 2003*.