Optimally Efficient Multicast in Structured Peer-to-Peer Networks

Dirk Bradler* Jussi Kangasharju† Max Mühlhäuser*
*University of Technology Darmstadt, Darmstadt, Germany
†University of Helsinki, Helsinki, Finland

***Abstract*—The Distributed Tree Construction (DTC) algorithm is designed for optimally efficient multicast tree construction over structured peer-to-peer networks. It achieves this by creating a spanning tree over the peers in the multicast group, using only information available locally on each peer. Furthermore, we show that the tree depth has the same upper bound as a regular DHT lookup which in turn guarantees fast and responsive runtime behavior. Our DTC algorithm is DHT-agnostic and works with most existing DHTs. We evaluate the performance of DTC over several DHTs by comparing the performance to existing application-level multicast solutions, we show that DTC sends 30–250% fewer messages than common solutions.**

I. Introduction

Structured peer-to-peer networks (a.k.a. distributed hash tables, DHT) are peer-to-peer overlays where the structuring mechanism (typically one or more hash functions) uniquely determine the location of a peer in the overlay and its neighbors, as well as the placement of content on peers. Application level multicast solutions (ALM) also build an overlay of the nodes participating in the multicast. This overlay can have any structure, but tree is a very common choice, especially for 1-to-n multicast. This scenario is our focus in this paper. Our contribution is a distributed tree construction (DTC) algorithm. The key feature of our algorithm is that it creates a spanning tree *without any communication between the peers*, using only local information available to every participating peer. Because it is a tree, we are guaranteed that only the minimum number of messages needs to be sent during multicast. These key features set our algorithm apart from previous work, such as the application-level multicast proposed by Ratnasamy et al. [1] or SplitStream [2]. Previous work typically either has a high number of duplicate messages, requires additional coordination traffic or needs a second overlay network. Additionally range and origin of the spanning tree can freely be set to any value and any peer.

Our DTC algorithm presented in Section II has none of the shortcomings of the previous algorithms. It can be used on top of most existing DHTs and it works using information available at each peer about the peer's neighbors. As we show in Section II, this standard information maintained by the DHT is sufficient for spanning a tree. Our DTC algorithm yields optimal performance in terms of messages sent. We also show that the overhead of duplicate messages in existing solutions is at least 30% but can in several cases be up to 250%.

This paper is organized as follows. In Section II, we present how we construct the distributed spanning tree and prove its properties. In Section III we present how multicast can be implemented with DTC. Section IV evaluates our algorithm on different DHTs and compares its performance against existing algorithms. In Section V, we discuss the robustness of our algorithm and present mechanisms for improving its resilience against malicious peers. Section VI discusses related work. Finally, Section VII concludes the paper.

II. Distributed Tree Construction

We build our DTC algorithm on top of a structured peer-to-peer overlay. We first discuss the requirements on the structured overlay, and then present the DTC algorithm with optimality proofs. The idea behind DTC is to build a spanning tree to connect all the nodes in the multicast tree. When a message is sent from the root, every node in the tree receives it exactly once. The challenge lies in constructing the tree without any overhead and using only local information available in each node.

A. Structured Overlay

Our DTC algorithm works on any structured overlay which fulfills the property that every node knows *all* of its immediate neighbors in the overlay hash space. Networks like Chord [3], CAN [4] and VoroNet [5] obviously fulfill this property. In case of Chord, the critical information is knowing the successor and for CAN, knowing all neighbors in all coordinate directions. In case of Pastry [6] the condition is fulfilled since the leaf sets of all nodes always contain the closest neighbors in the hash space. Although Tapestry [7] is very similar to Pastry, it does not have the equivalent of the leaf set and thus might not be suitable without modifications. Kademlia [8] also fulfills the required condition, since the buckets for the shorter distances contain the closest nodes in the hash space.

In the remainder of this paper, we consider only Chord and CAN as overlay networks. Although the principle of DTC in both networks is the same, differences in the overlay structures lead to performance differences (see Section IV).

B. DTC Algorithm

We span the tree from a point in the overlay according to the overlay routing. The area of the overlay which the spanning tree is supposed to cover is explicitly defined. The information about the root of the tree and the area are sufficient to construct the spanning tree in a purely distributed manner. We will now show how this can be done in Chord and CAN.

978-1-4244-2309-5/09/$25.00 ©2009 IEEE

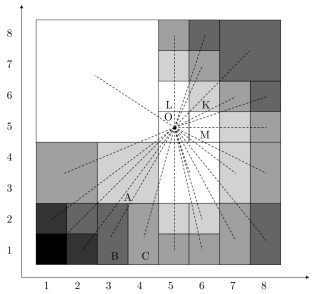


Fig. 1. Example spanning tree of a CAN

Note that even though the algorithm constructs a spanning tree, no peer has a complete view of the tree. The tree is always constructed on-demand by having the root send a message which constructs the tree as it gets passed through the peers. Thus, when we say below that a peer adds some other peers to the tree as its children, it means in practice that the peer in question forwards a message to the other peers.

Example: Chord

In case of Chord, the area is an arc on the Chord ring and the root of the tree is the first node on the arc. The root of the tree selects all of its fingers that are in the area as its children. Each of them will recursively perform the same operation until all peers in the area (the arc) have been included in the tree. Any of the peers can determine which fingers it should include, since it knows the root and the length of the arc.

Example: CAN

In case of CAN, the area is a convex area of the d-dimensional coordinate space, with the root somewhere in this area. The restriction to convex areas is imposed by the algorithm. Non-convex areas can be used by splitting the area into non-overlapping convex areas which cover the desired area. The root first adds its immediate neighbors ($2d$ neighbors in a d-dimensional CAN), which then continue adding their neighbors, according to the rules defined below. As in the Chord-case above, the information about the root of the tree and the area it is supposed to cover are available to the peers. The area can be defined either with simply the radius of the area, or by specifying for each dimension separately how far the area reaches in that dimension; the only restriction on the area is that it must be convex.

We assume every node knows the following:

- Size of the zone of each neighbor (maintained by standard CAN routines)
- Root of the tree and the area it is supposed to cover (available in the message which is used to create the tree)

Figure 1 shows a spanning tree of a two dimensional CAN. The tree is rooted at the white zone marked O at coordinates (5, 5). The other white zones are children of the root, and the levels of the tree are shown in darkening shades of gray.

The tree is constructed as follows. When a peer X receives the message, it computes for all of its neighbors the vector from the center of the root's zone to the center of the neighbor's zone. If that vector intersects the common border surface between X and the neighbor, then X adds that neighbor as its child. Consider the third zone from the left on the bottom row in Figure 1, marked B. It has neighbor A on the top and neighbor C to the right. The vector from the root passes through both of these neighbors, but the one on the right (marked C) is the parent of node B.

It is important to note that every node is able to compute the vectors and determine whether it should add any of its neighbors as children using only information available locally through normal overlay communications. No coordination between nodes is needed, nor is any additional traffic generated.

In some cases, it is possible that the vector between the root and a zone Z does not pass through any direct CAN neighbor of Z. For example, in Figure 1, the vector between the root and the zone marked K passes directly through the corner point of the two zones. Depending on how the ownership of edges is defined, it is possible that there is no neighbor through whose zone the vector passes on its way from root to K. (Note that regardless of how the ownership of edges is defined this problem persists.) In general, this issue arises when two zones share up to $(d-2)$ dimensions in a d-dimensional CAN (e.g., a point in 2-dimensional CAN and a point or a line in a 3-dimensional CAN). In this case, the forwarding algorithm does not reach all nodes. We have defined the following tie breaker.

The Tie Breaker: We use the following rule for determining how to construct the tree in the above case. The two problematic zones differ in at least 2 and up to d dimensions. We order the dimensions beforehand. The forwarding path should be such that the smallest dimensions with differences are used first. The length of the tie breaker path will be the same as the number of dimensions in which the two problem zones differ (i.e., between 2 and d). Note that none of the nodes on the path would normally forward the message, but *all of them* are able to compute locally that they are part of the tie breaker procedure and are able to perform their duties correctly. In the example of Figure 1, the tie breaker would mean that M is the node responsible for adding K as its child, since x-coordinate is considered before y-coordinate.

C. Proof of Optimality

We prove the following properties of our DTC algorithm in [9]. For space reasons, we omit the proofs in this paper.

1) The DTC algorithm creates a spanning tree over the area
2) The depth of the tree is proportional to message complexity of the underlying DHT

III. Multicast with DTC

With DTC, we can create a spanning tree over any area of a suitable DHT and use the tree to send a message to all nodes in the area with no overhead. There are two natural ways of implementing multicast with DTC. One solution is to create a new DHT for every multicast group, as proposed in [1]. This solution isolates the multicast groups from each other and requires that any peer wanting to participate in multiple groups must join multiple DHTs and participate in the management.

Another solution is to have only one DHT and assign the multicast groups to different areas of the DHT. If several peers

Messages Received	DTC CAN	Simple Flooding	ALM	DTC Chord
0-1	2000	1	1143	2000
2-3	0	2	684	0
4-5	0	6	78	0
6-7	0	22	12	0
8-9	0	67	2	0
10-11	0	247	0	0
12-13	0	925	0	0
≥ 14	0	7851	0	0
Sum	2000	26106	3087	2000

TABLE I
AVERAGE NUMBER OF MESSAGES RECEIVED PER NODE

are members of several multicast groups, it is possible to assign the groups near each other, so that each peer only has to join the DHT once. (The assignment requires some additional coordination, and may need to be revised from time to time.) This can be a major advantage in case the actual content is split over several multicast groups, e.g., [2], [10], so that clients can choose the level of quality they desire. Increasing levels of quality would represent larger and larger areas in the DHT.

Both solutions work well in practice and the DTC algorithm works well on both kinds of networks.

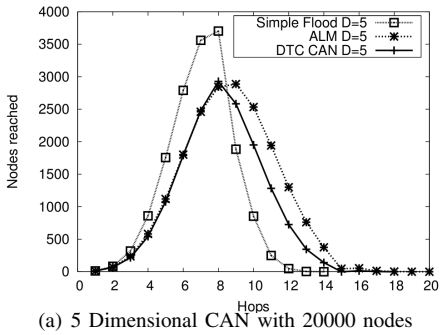
IV. EVALUATION

We now evaluate the performance of our DTC algorithm and compare it against two other solutions. We tested DTC over two DHTs, Chord and CAN, to show how its performance in some cases depends on the underlying DHT. As comparison, we have selected the application-level multicast on CAN by Ratnasamy et al. [1] and a simple flooding scheme, where each node just forwards a message to all of its neighbors, except the one where it got the message from.

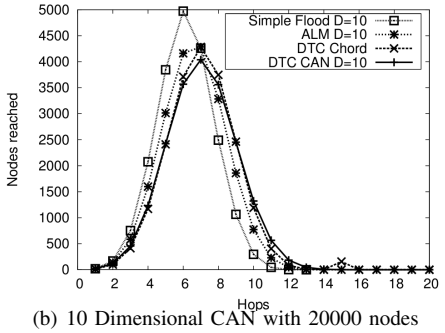
We used the PlanetSim P2P simulator [11]. Jones et al. [12] report some issues in CAN implementations, which we took into account. We verified our CAN implementation by comparing it to the results in [12] and thorough testing on our own. Our Chord implementation is directly based on the original work in [3]. Our implementation of the ALM algorithm of [1] did not face the race-condition mentioned in [12], [13], since PlanetSim is cycle-based, which prevents the race-condition.

We varied the size of the network and also the number of dimensions in the CAN. Simulations were repeated 30 times and the reported numbers are averages over the 30 runs. We measured the number of messages received by each node and the depth of the spanning tree. The first metric determines how (in)efficient the mechanism is and the second determines how quickly all the peers receive the message.

Table I shows how many messages a given node received in a 2000 node CAN with 10 dimensions. (DTC-Chord was run on a standard Chord of 2000 nodes.) The table shows how many nodes on average received the message from the root. We cut the table at 14 messages per node and summed up all the nodes that received the message 14 times or more (applies only to simple flooding). The last row shows the total number of messages sent in the system. Since there are 2000 nodes, 2000 messages are sufficient in the optimal case.



(a) 5 Dimensional CAN with 20000 nodes



(b) 10 Dimensional CAN with 20000 nodes

Fig. 2. Tree depth

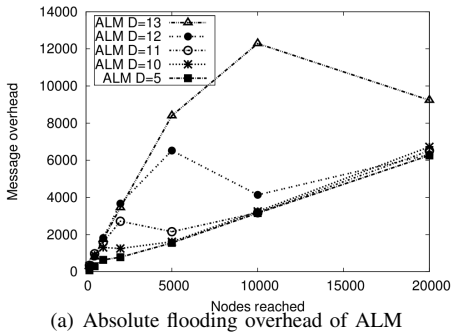
The two DTC-based solutions do not generate any duplicate messages. ALM performs relatively well, generating about 50% too many messages. We return to the evaluation of the overhead of ALM below. As Table I shows, simple flooding has a high overhead in terms of messages sent. The shown overhead is typical of the performance of simple flooding.

The performance of the DTC-based algorithms was as expected in all investigated parameter combinations. Both of them were able to perform their task always with the *minimum* number of messages, i.e., as many messages as nodes.

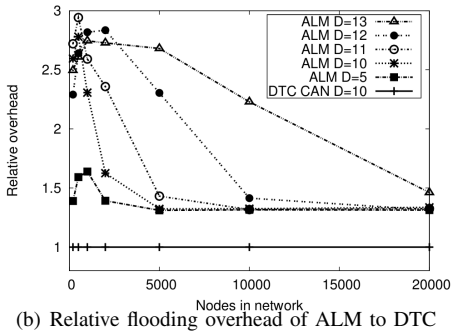
We also evaluated the depth of the spanning tree, since this directly affects the time it takes to complete the multicast. We compared different network sizes from 200 to 20000 nodes and different dimensions in CAN (ranging from 2 to 20). Figure 2 shows 5- and 10-dimensional CANs with 20000 nodes. In Figure 2(b) we also plot DTC-Chord. The x-axis shows the number of hops and the y-axis shows how many nodes are at that depth in the spanning tree.

Simple flooding has the smallest depth because it always takes the shortest path to each node. Therefore all nodes are reached with the minimum number of hops. DTC-CAN and ALM have performance which is slightly lower than simple flooding and are very close to each other. In the case of the 5-dimensional network and 20000 nodes the optimum would be about 7 hops, while ALM needs 9 and DTC-CAN 8 hops in the average. The greater the number of dimensions used for the CAN network, the smaller is the difference between the approaches. Differences in the 10-dimensional CAN network (see figure 2(b)) with 20000 nodes are almost non-existent; both ALM and DTC-CAN need about 7-8 hops on an average, while the optimal case would be about 6 hops.

DTC-Chord (shown only in Figure 2(b)) has performance



(a) Absolute flooding overhead of ALM



(b) Relative flooding overhead of ALM to DTC

Fig. 3. Flooding overheads

similar to DTC-CAN and ALM in the 10-dimensional case. The finger tables of Chord reduce the number of needed hops effectively. While the CAN network is able to be further optimized by using more dimensions, the number of hops for Chord is proportional to the density of the finger tables.

We investigated the performance with dimensions ranging up to 20, but we did not observe any significant improvement in performance of DTC-CAN or ALM after 10 dimensions.

We now turn to evaluating the overhead of ALM, which was already shown in Table I. We compare ALM against DTC-CAN. Figures 3(a) and 3(b) show how the overhead evolves as function of network size. The x-axis shows the number of simulated nodes; we started from 200 nodes and simulated up to 20000 nodes. The y-axis shows the average number of generated messages. We show several variants of ALM, each with different number of dimensions. Note that DTC-CAN was always able to perform optimally.

Figure 3(a) shows the absolute overhead, i.e., how many unnecessary messages ALM sent and Figure 3(b) shows the relative overhead compared to DTC-CAN (DTC-CAN has overhead 1). As the network grows, the relative overhead of ALM tends to about 32%. However, Figure 3(a) shows the interesting behavior of ALM. For every number of dimensions, the curve has several sharp corners where the overhead changes considerably. The reason for this is as follows.

The message overhead is influenced by the number of dimensions and nodes. The more dimensions, the more nodes are needed to populate the d-dimensional ID space uniformly. As soon as $\sqrt[d]{n} \geq 2$, the overhead starts to converge to the observed 32%. Below this threshold, the number of duplicate messages steadily increases with more nodes. This explains the high peaks for the smaller networks in Figures 3(a) and as Figure 3(b) shows, the resulting overhead can be up to 250%.

In summary, the DTC-based approaches generate only the minimum amount of traffic, while keeping the depth of the spanning tree similar to ALM. ALM has a message overhead of at least 32%, in many cases up to 250%. Simple flooding has an extremely high overhead.

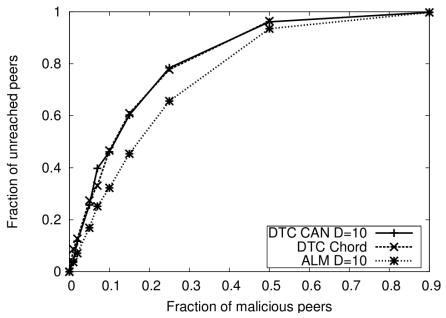


Fig. 4. Impact of malicious peers

V. Robustness of DTC

Because the DTC algorithm (and similar approaches [1], [2]) builds a tree, it is vulnerable to peers crashing or behaving maliciously. If a peer does not forward the message, then the sub-tree rooted at it will not be included in the spanning tree.

Because all DHTs have mechanisms to detect crashed peers and recover, we do not consider such system failures to be a problem. The only case a message could be lost is if a peer crashes between receiving a message and forwarding it, which is extremely rare. In all other cases, we assume that the standard DHT maintenance takes precedence over the tree spanning messages and that in such cases, the overlay will first be healed and thus no message loss or duplication can occur.

In the following, we consider the case of a malicious peer not forwarding the message onwards. We evaluated the severity of this problem by spanning a tree over a randomly generated 20000-peer network. The CAN networks had 10 dimensions. We varied the fraction of malicious peers and selected the malicious peers uniformly at random. For each case, we measured the number of peers who were not part of the tree. Each parameter combination was repeated 30 times and the results we present are averaged over all the runs.

Figure 4 shows the fraction of unreached peers as a function of malicious peers for DTC-CAN, DTC-Chord, and ALM. 10% of malicious peers are able to cut off on average 20% of the peers from the spanning tree. Because the DTC algorithm eliminates all duplicate messages, it is particularly vulnerable to malicious peers, but the ALM does not fare much better. Only when the fraction of malicious peers is between 10% and 50% does ALM have any marked improvement over DTC-based systems. Although not shown on Figure 4, the simple flooding approach is extremely resistant against malicious peers. Peers become unreached only when the fraction of malicious peers is very high (typically over 70–80%).

We made an observation about the effects of the underlying DHT on the performance of DTC-algorithms. In a CAN with many dimensions, even if a malicious node is near the root in

the spanning tree, it cannot do much damage. In contrast, if the longest finger of the root in Chord is malicious, it is able to cut half of the spanning tree. Thus, a Chord-based DTC is slightly more vulnerable than a CAN-based DTC.

We now propose a solution for remedying the problems caused by malicious peers. Our primary goal is *detecting* the presence of malicious peers, with as little overhead as possible.

In multicast there is no guaranteed feedback channel from the receivers to the root. One possibility is for the root to require acknowledgments from all nodes. In the interest of scalability, the acknowledgments should be propagated back along the tree. Furthermore, each peer should sign its response with a key that is tied to its zone of responsibility in the DHT and its IP address. These signatures increase the likelihood of discovering malicious peers, because the malicious peer would have to invent the zones of responsibility and IP addresses for all the peers in its subtree. These might overlap with legitimate zones from other parts of the tree, thus indicating the presence of a malicious peer. Furthermore, the root could also verify some of the leafs of the tree (using additional communications) that the responses were actually sent by them. Although this would reduce the effects of malicious peers, in the absence of a centralized identity management scheme, it cannot completely eliminate them. A further refinement to this technique is to require acknowledgments from only some of the messages, by flagging them. The flagged messages should also contain information about the non-flagged messages, to avoid the problem of malicious nodes only forwarding the flagged packets.

Note that the acknowledgment scheme does generate additional traffic. However, without such a mechanism we have no means of detecting malicious peers in the network.

VI. Related Work

Flooding approaches for P2P networks in general have been extensively investigated [14]–[16]. While simple flooding approaches may apply to unstructured networks, DHT networks can take advantage of the structured neighbor lists and reduce the number of duplicate messages. This was already done in the Pastry overlay network [17].

Ratnasamy et al. [1] defined an application-level multicast on top of CAN. As our evaluation shows, this approach can have a significant overhead and even in the best case, will have an overhead of about 32%. As we have shown, the overhead is a function of the network size and CAN dimensions and in smaller areas, the overhead can grow considerably higher. An improved version of ALM in [13] reduces duplicate messages, but duplicates may still occur, especially in the case of uneven zone sizes within the CAN overlay.

Several works on peer-to-peer-based application-level multicast, e.g., [2], [10], [18], propose splitting large multicast streams into several sub-streams and clients can then choose what they want to receive. Usually the sub-streams relate to the quality of the video. These techniques are complementary to our DTC. We can split the content into different multicast groups and construct the trees with DTC. In this case, it is highly beneficial to have multiple multicast groups in a single DHT, as discussed in Section III.

VII. Conclusion

In this paper we have presented a distributed tree construction algorithm, which is able to create a spanning tree over a part of a DHT, with no inter-node communication and using only information available locally on each node. Our evaluation shows that DTC performs as expected and the comparison to similar approaches from literature shows that the message overhead of existing solutions is considerably higher than the optimal number of messages sent by DTC. The depth of the spanning trees are similar in all studied cases.

References

[1] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Application-level multicast using content-addressable networks," *Proceedings of NGC*, 2001.

[2] M. Castro et al., "SplitStream: High-bandwidth multicast in a cooperative environment," in *Proceedings of ACM Symposium on Operating Systems Principles*, Lake Bolton, NY, Oct. 2003.

[3] I. Stoica et al., "Chord: A scalable peer-to-peer lookup service for Internet applications," in *Proceedings of ACM SIGCOMM*, San Diego, CA, Aug. 2001.

[4] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proceedings of ACM SIGCOMM*, San Diego, CA, Aug. 2001.

[5] O. Beaumont, A.M. Kermarrec, L. Marchal, and E. Riviere, "VoroNet: A scalable object network based on Voronoi tessellations," *IEEE IPDPS*, pp. 1–10, 2007.

[6] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, Nov. 2001.

[7] B. Y. Zhao et al., "Tapestry: A resilient global-scale overlay for service deployment," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, pp. 41–53, Jan. 2004.

[8] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *Proceedings of International Workshop on Peer-to-Peer Systems*, Cambridge, MA, Mar. 2002.

[9] D. Bradler, J. Kangasharju, and M. Mühlhäuser, "Optimally efficient prefix search and multicast in structured peer-to-peer networks," Tech. Rep. TUD-CS-2008-103, Department of Computer Science, Darmstadt University of Technology, Aug. 2008.

[10] N. Magharei and R. Rejaie, "PRIME: Peer-to-Peer Receiver-drIven MEsh-based Streaming," in *Proceedings of IEEE Infocom*, 2007.

[11] P. García et al., "PlanetSim: A New Overlay Network Simulation Framework," *Lecture Notes in Computer Science (LNCS), Software Engineering and Middleware (SEM)*, vol. 3437, pp. 123–137, 2005.

[12] M.B. Jones, M. Theimer, H. Wang, and A. Wolman, "Unexpected complexity: Experiences tuning and extending CAN," *Microsoft Research, Tech. Rep. MSR-TR-2002-118*, 2002.

[13] M. Castro et al., "An evaluation of scalable application-level multicast built using peer-to-peer overlays," vol. 2, pp. 1510–1520, March 2003.

[14] S. Jiang, L. Guo, and X. Zhang, "LightFlood: an efficient flooding scheme for file search in unstructured peer-to-peer systems," *in International Conference Parallel Processing*, pp. 627–635, 2003.

[15] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making gnutella-like P2P systems scalable," *SIGCOMM*, 2003.

[16] W. W. Terpstra, J. Kangasharju, C. Leng, and A. P. Buchmann, "BubbleStorm: Resilient, probabilistic, and exhaustive peer-to-peer search," in *Proceedings of ACM SIGCOMM*, Kyoto, Japan, Aug. 2007.

[17] V. Vishnevsky, A. Safonov, M. Yakimov, E. Shim, and A.D. Gelman, "Scalable blind search and broadcasting over Distributed Hash Tables," *Computer Communications*, vol. 31, no. 2, pp. 292–303, 2008.

[18] N. Magharei, R. Rejaie, and Y. Guo, "Mesh or Multiple-Tree: A Comparative Study of Live P2P Streaming Approaches," in *Proceedings of IEEE Infocom*, 2007.