Multicast Tree Reconfiguration in Distributed Interactive Applications

Carsten Griwodz[1,2], Knut-Helge Vik[1], Pål Halvorsen[1,2]
[1]IFI, University of Oslo, Norway   [2]Simula Research Laboratory, Norway
Email: Email: {griff, knuthelv, paalh}@ifi.uio.no

Abstract— **Communication in highly interactive distributed applications, such as massive multiplayer online games, can often be performed efficiently using multicast, i.e., application level multicast. However, in applications with a very dynamic group management, the multicast tree will have frequent changes, and in applications that have stringent latency requirement, this operation needs to be fast. Current multicast approaches either have no notion of reconfiguration, they do not care about tree reconstruction latency or wrongly assume that this is a fast, atomic operation. In this paper, we have focused on dynamic reconfiguration and have tested different ways for a node to join a tree. Our results show that this is an important issue for the class of highly interactive distributed applications.**

I. Introduction

Large improvements in computer technology have enabled highly interactive distributed applications such as distributed virtual environments, massive multiplayer online games (MMOGs) and computer supported cooperative work. These applications often include several types of media ranging from text to continuous media and may have very stringent requirements with respect to the quality of the client data playout.

In the MiSMoSS project, we aim for better system support for such applications, i.e., trying to make more efficient use of the available resources by offering mechanisms like prediction, area-of-interest (AoI) management, group communication, aggregation, replication, etc. In particular, we look at MMOGs due to the mix of different media, the stringent latency requirements, the dynamic client groups and the fact that it has become a popular, fast growing, multi-million industry with a very high user mass. Today, MMOGs are increasing in size and complexity, supporting hundreds or thousands of concurrent players [1], and they typically include a mixture of game situations from role-playing games, first person shooter games and real-time strategy games. Players in the game move around and interact with other players, seemingly as if they were located next to each other. Frequently, many players interact with each other and the same object in the game world, these are then said to share an AoI or to be within each others AoI. If the game is large enough, the majority of players will not share an AoIs.

Today, most MMOGs apply a central server approach for collecting and processing game events generated by players, and point-to-point communication for the distribution of game state updates. They have few, if any, mechanisms to optimize event distribution. This approach is applicable because games are designed with extremely low bandwidth for players' access networks in mind. Even these thin individual data streams contribute to game server scalability problems if the number of players that share an AoI is high, forcing game designers to prevent game situations that can lead to such clustering. It would therefore be desirable to distribute the game events efficiently, such that 1) the perceived game quality is above a satisfactory threshold regardless of system capacity and geographical distance and 2) the available resources are utilized efficiently. By grouping players according to their AoI, group communication may be an efficient means to achieve this. Lacking working network layer multicast in the Internet, we consider application layer multicast (ALM) and investigate group communication algorithms appropriate for MMOGs to enable efficient event distribution.

In this paper, we focus on the cost of multicast tree reconfiguration. There is a lot of existing work on multicast group maintenance. Both online (dynamic trees) and offline (static trees) algorithms exist, but only online algorithms are applicable, since players' AoI changes frequently. In fact, AoIs in a modern game can be entirely player-specific and have an adaptive size (as opposed to, e.g., room-sized AoIs), which implies that the membership of a group interacting with a particular object can be extremely volatile. Current online mechanisms do not consider tree reconstruction latency or wrongly assume that this is a fast atomic operation. We have therefore tested several ways of reconfiguring (joining) a tree and show that this is an important issue for the class of highly interactive distributed applications. The approaches tested do not yet address the problem of non-trivial leave operations where three or more neighbors are involved. We share this problem with much of the existing work, and recognize from the presented results that this is problematic.

II. Application Level Multicast

An MMOG may have players located all around the world. Thus, choosing a suitable group communication style is important, both in terms of improving the service to the users and optimizing the resource utilization of the system. Multicast provides efficient means for group communication where the groups are typically organized in a tree structure. One implementation of multicast is IP multicast, but it is not fully deployed in the Internet and lacks features like address filtering and group membership control. The alternative is ALM. Approaches add group membership control, and make

it easy to support high level functionalities. Compared to IP multicast, ALM is necessarily less efficient in terms of latency but is easier to deploy.

In highly interactive distributed applications, especially MMOGs, each user frequently both sends and receives data from other users in the same group. In this scenario, we have three important, but contradictory, challenges:

1) The maximum delay between any pair of nodes must not exceed a given threshold.
2) The total overall cost of transmitting data in the multicast tree should be minimized.
3) Multicast tree reconfiguration must be supported online, and this operation must be fast.

In literature, a lot of different algorithms have been proposed, but current multicast approaches have insufficient support for *fast* reconfiguration which is required in an MMOG scenario due to for example players joining and leaving a group (entering another room), links becoming overloaded (too many players sharing a link), nodes going down (a player quits), reaching maximum tree node-to-node latency (increased latency on a link due to congestion), etc. As a first step towards an appropriate multicast algorithm in the MMOG scenario, we therefore look at the cost of online multicast tree reconfiguration. We compare approaches for performing join operations, and are also evaluating tree cost if no additional reconfiguration is performed.

III. RELATED WORK

The construction of low-cost, delay-bounded multicast trees has been investigated several times before, often as Steiner tree heuristics. However, to the best of our knowledge, existing mechanisms do not meet all of the required properties from section II. Recently, some delay bounded, many-to-many multicast algorithms are proposed [2], [3], but these (as many one-to-many approaches) assume scenarios where all group members start the session at the same time.

Online multicast algorithms support dynamic tree manipulations [4]–[9]. Typical operations include join and leave, and some allow online rearrangement of the multicast tree. A join operation is typically performed using the shortest path [10], [11] or the delay constrained minimum cost path [12]–[14]. This is cheaper than tearing down the tree and re-building it from scratch, but it will probably give a larger cost increase of the tree. A remove operation deletes a node from the multicast tree. A leaf node is trivially removed. Deleting a node with a degree of 2 results in two subtrees that for example are connected using the least cost path [15] or least cost delay bound path [13]. If the node has a larger degree (≥ 3), the node is often kept in the tree for routing purposes, an approach that we follow as well. Additionally, to reduce the cost of the tree, some algorithms allow periodic tree rearrangement. The cost of the tree may be periodically calculated in the background, and a complete rearrangement operation may be triggered based on some threshold value. It has been proposed to divide the tree into regions where the quality of the region is associated with the number of changes that have been made without re-optimizing the region [10], [12].

Existing approaches assume that reconfiguration operations are atomic, i.e., that new join and remove operations do not occur before previous rearrangement has been completed. MMOGs do not allow such assumptions. It is critical that tree reconfiguration is fast, in particular node join operations as new nodes should receive the data on time. It is also vital that leave operations keep the multicast tree intact for all remaining nodes. Rearranging the tree while the data is flowing may cause members to loose data if the operation is not handled appropriately. Existing interactive applications such as ACTIVE [16] perform join operations more quickly, but the performance of the operation has not been given particular consideration, either.

IV. SIMULATION AND RESULTS

To understand the time that is consumed by different approaches for quickly joining a multicast group, and the worst-case delays in the resulting overlay networks, we use simulation. In our simulation, we distinguish nodes that represent Internet routers (the backbone network) and nodes that represent computers participating the distributed game (clients and servers).

We use the topology generator BRITE [17] to create backbone networks as graphs with 1000 nodes. The link delay between neighboring nodes is uniformly distributed between 2 ms and 50 ms. The servers and clients are placed in the network by attaching them randomly as leaf nodes to the backbone network. One to three leaf nodes are connected to each of the backbone nodes, and one of them is manually selected as the game server. This is meant to model that the game is deployed independently from any network provider. We don't assume support for multicast either. Since we for now are only concerned with group maintenance and not data exchange, we ignore bandwidth in this simulation. To compare topological effects, we have run all simulations with the backbones nodes arranged in both flat and hierarchical topologies. The hierarchical topologies organize them 10 networks of 100 nodes, respectively.

A. Group membership

Group membership in our simulations is not modeled after the behavior of any actual applications, where nodes are usually active members of a group for a longer while, and where they are members of several groups at the same time. Rather than that, our simulated clients remain members of a group within the game only for a very short time, and change to another group afterwards. This time is short with respect to the maximum end-to-end delay in our topologies, which is approximately 800 ms, a value that is taken from real-world MMOG traces. We do this to investigate how well join algorithms perform when group membership is frequently outdated.

The central server is not member of any group. It provides a lookup service that allows clients entering a group to locate

(a) connect first with invitation   (b) connect best with invitation   (c) connect first without invitation   (d) connect to closest address
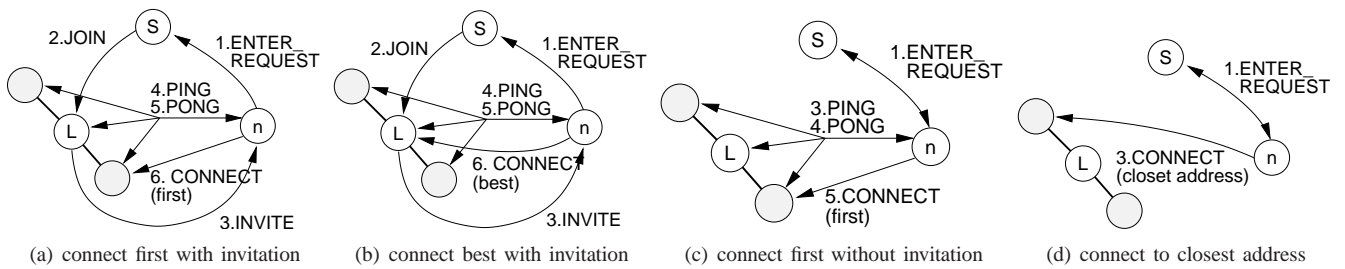
Fig. 1. Protocols variations

group members. Every group has a leader, which is the node that has the smallest maximum delay to all other group members. Whenever a group leader determines that one of its neighbors has a smaller maximum delay, it hands over group leadership. The new leader informs the central server of the change.

To investigate the performance of the algorithms both in equally sized and heavily unequally sized groups, we create as many empty groups as the simulation has client nodes. To achieve equal group sizes, we let nodes perform a random walk between nodes, where those are arranged in a square. To achieve unequally sized groups, we assign groups Zipf-distributed popularities.

B. Protocol variations

Figure 1 shows the approaches for joining groups that we have investigated in this work. The letter S marks the server, L the group leader, and n the client that tries to newly join a group. In all cases, a node that joins a group contacts a server first. However, this membership and leadership information will be outdated frequently because average membership duration is short with respect to the end-to-end delay in the system.

In the scenario *connect first with invitation* (figure 1(a)), the server is notifying the group leader of the intention to join, which will then send on invitation to the new node that includes the addresses of all group members. The new node contacts all of those nodes (PING). As soon as it receives an answer (PONG) from one of the nodes, it connects itself to the overlay network formed by the group members at that node. We expect that this approach would create deep graphs with a high worst-case delay inside groups but fast join operations.

The scenario *connect best with invitation* (figure 1(b)) is nearly the same, but the joining node waits for answers from all contacts nodes until it chooses to connect to the node that has the lowest worst-case latency to other group members. We expect that this approach would create wide graphs with slow join operations but a low worst-case delay inside groups.

The scenario *connect first without invitation* (figure 1(c)) resembles the first scenario, but the server is sending group membership information to the joining node, instead of delegating that operation to the group leader. While this avoids the latency that is added by contacting the group leader first, the group membership information at the server is usually less up-to-date than the group leaders, which may make it necessary to retry the join operation more frequently.

The scenario *connect to closest address* (figure 1(d)) is the simplest one. It is similar to the previous approach, but the joining node uses the network address to estimate physical distance, and connects to the node with the most similar one. We expect very fast join operations but a rather high worst-case delay inside groups from this approach.

No node in any of these approaches collects enough topological information anywhere to be able to compute an optimal tree for the group. In fact, nodes know only their neighbors and the worst case delays achieved through these neighbors - all other information becomes outdated too quickly because membership duration is frequently shorter than the maximum delay between pairs of group members.

C. Evaluation

We have compared the four variations of our protocol by simulation. To evaluate them, we use *join latency* and the *tree cost* as performance metrics. Join latency is an issue according to challenge 3 in section II, i.e., how fast a player can resume the game-play after the AoI has been changed such that a group membership change is required. Tree cost addresses challenge 2, i.e., the total cost in terms of the sum of all edge delays indicating the overall system performance. The figures shown here use networks with 1000 nodes. Other network sizes show the same trends, but more sparsely populated networks do not show particular changes for the most popular groups.

The time that a node stays in a group is only partly relevant for efficiency of the multicast tree that is created according to our algorithm. In general, all of our approaches will lead to the same multicast tree on all time scales. However, in case of long end-to-end delays and short membership duration, the contact information that a joining node must retrieve from a central entity (S in figure 1) will frequently be outdated. Similarly, no single node in a multicast tree can safely know all other group members at a given time. To provoke this situation, we have chosen 100 ms for our simulation. This is a frequently used update frequency in central server games, which limits also the frequency of dynamic AoI changes.

In the figures, we show the results for Zipf-distributed group popularities: a small X value in the graph represents a large groups, while a large X value represents a small group. The graphs in figure 2 show the percentage of the time in a group
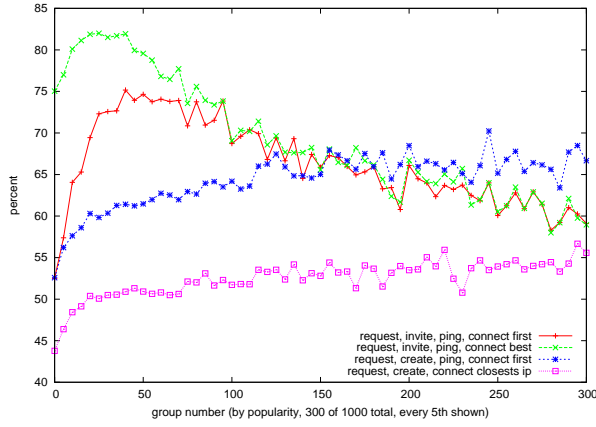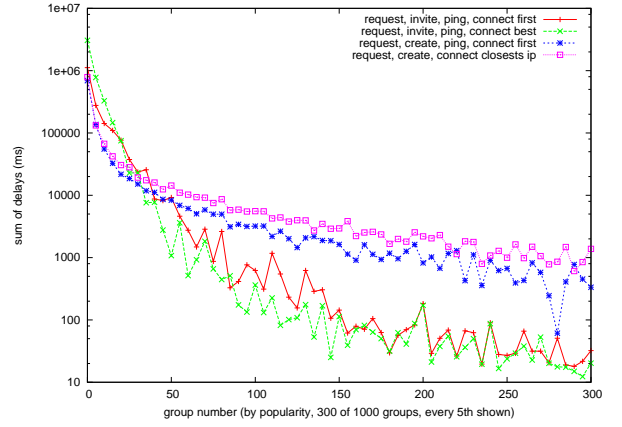
(a) Autonomous system
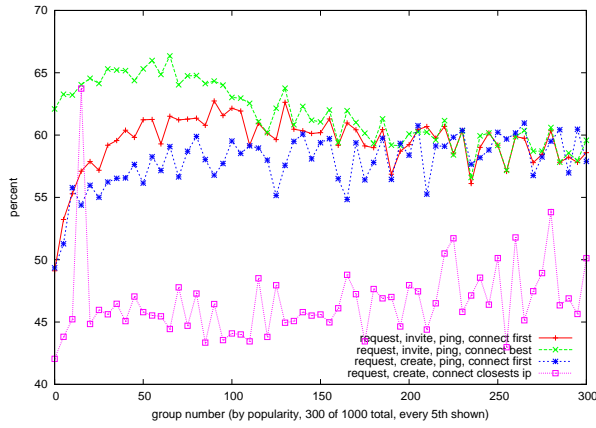
(b) Flat network

Fig. 2.   Percentage of time spent entering a group, average of 10 runs

(a) Autonomous system

(b) Flat network

Fig. 3.   Sum of all edge delays in a group, average of 10 runs

that is used performing the steps described in section IV-B. Figure 3 shows the average cost of the overlay network connecting all nodes in a group, where the cost is the sum of all link delays.
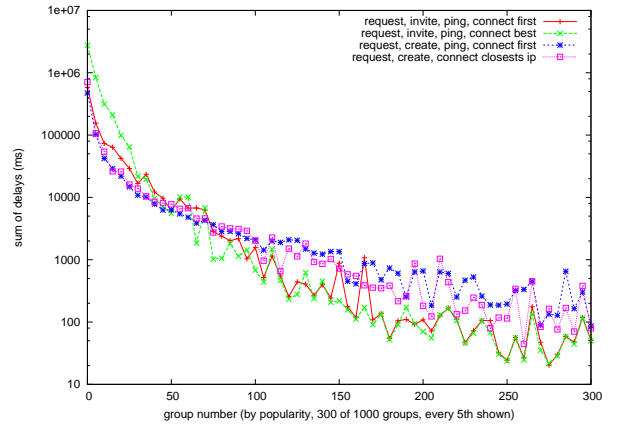
1) Join time: As expected, the approach that joins the node that is closest by its network level address spends the highest percentage of its time actually in the group, while the other approaches spend more time performing the steps of figure 1. Even for the case of groups that are unpopular and therefore frequently found empty, this is still true. In general, however, the algorithms perform similarly for those nodes because nearly all time is consumed in contacting the central server for the initial reference to the group.

It is more remarkable that the more complex joining approaches are also nearly as fast as connecting to closest address when groups have a large number of members. The reason is that the joining node is offered many nodes to connect to, some of which are bound to be close to it. The connect step itself is therefore fast.

For the groups with a medium number of nodes, on the other hand, connect first/best with invitation shows large delays. The problem here is that join operations are likely to trigger a leader change in the group. Since the central server sends

invitations only to the group leader, these are frequently refused because the central server's information is already outdated. This, in turn, requires that the central server sends another invitation to the new leader, which is time-consuming.

2) Group cost: Connect best with invitation achieves consistently the cheapest graphs (in terms of the sum of all unidirectional delays) when all groups have the same popularity. The created graphs are also cheapest in the Zipf scenario when groups have medium or low popularity.

However, when the group popularity is high, this is not the case anymore. The reason for this is that we use the same leave operation that is described in section III, ie. we do not remove a leaving node from the multicast tree before the number of its direct neighbours drop below three. Connect with invitation leads to wide graphs, while connect without invitation leads to deep graphs. Wide graphs are bound to have many nodes with more than three neighbors. Nodes close to the group leader can therefore rarely leave the tree when they decide to leave the group, and connections through them contribute to the overall cost of the tree.

3) Summary: Our results show that the time to join a multicast group can be significant and the chosen protocol can have large effects on the total cost. Additionally, note that

we only have simulated simple protocols looking at join time. If calculations to check for goals 1 and 2 in section II should be performed before joining, the join operation will consume considerably more time. However, the fast join operations lead to a degradation in tree costs for groups of medium group size. This implies that our further work on reconfiguration must support the removal of inner nodes of a tree when they leave groups.

V. CONCLUSION

In this paper, we have looked at the importance of reducing the multicast tree reconfiguration latency in highly interactive applications. We considered games in particular because there are example for central server games that benefit from extremely high group dynamicity, but we expect that an increasing number of distributed applications will require dynamic group membership. In this paper, we have compared several approaches for performing join operations, and performed leave operations only when the number of node neighbours is or drops below three. However, to give a complete view of this issue, several more tests must be performed, e.g., comparison with optimal tree costs, further protocol variations, network variations and cost factors. Our results indicate that the reconfiguration operation is a potential bottleneck for highly interactive applications. Our ongoing work includes a further investigation of online reconfiguration mechanisms (including group leave operations) in combination with different tree-algorithms, in particular Steiner tree algorithms, to be able to find a proper trade-off of the contradicting goals stated in section II.

REFERENCES

[1] Leo Sang-Min Whang and Jee Yeon Kim. The online game world as a product and the behavioral characteristics of online game consumers as role player. In *Proceedings of the Digital Games Research Association International Conference (DIGRA)*, June 2005.
[2] Chor Ping Low and Xueyan Song. On finding feasible solutions for the delay constrained group multicast routing problem. *IEEE Transactions on Computers*, 51(5):581–588, 2002.
[3] Hung-Ying Tyan, Jenifer C. Hou, and Bin Wang. Many-to-many multicast routing with temporal quality of service guarantees. *IEEE Journal of Selected Areas in Communications*, 52(6):826–832, 2003.
[4] H. Salama, Y. Viniotis, and D. Reeves. An efficient delay constrained minimum spanning tree heuristic. In *Proceedings of the 5th International Conference on Computer Communications and Networks (ICCCN)*, October 1996.
[5] George N. Rouskas and Ilia Baldine. Multicast routing with end-to-end delay and delay variation constraints. *IEEE Journal of Selected Areas in Communications*, 15(3):346–356, 1997.
[6] Xiaohua Jia. A distributed algorithm of delay-bounded multicast routing for multimedia applications in wide area networks. *IEEE/ACM Transactions on Networking*, 6(6):828–837, December 1998.
[7] Girish Kumar, Nishit Narang, and C. P. Ravikumar. Efficient algorithms for delay-bounded minimum cost path problem in communication networks. In *Proceedings of International Conference on High Performance Computing*, December 1998.
[8] Anna Hac and Kelei Zhou. A new heuristic algorithm for finding minimum-cost multicast trees with bounded path delay. *International Journal of Network Management*, 9:265–278, 1999.
[9] Lih-Chyau Wuu, Long Song Lin, and Shing-Chyi Shiao. Constructing delay-bounded multicast trees in computer networks. *Journal of Information Science and Enfineering*, 17(3):507–524, 2001.
[10] Fred Bauer and Anujan Varma. ARIES: A rearrangeable inexpensive edge-based on-line Steiner algorithm. In *Proceedings of the Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 361–368, 1996.
[11] Ehud Aharoni and Reuven Cohen. Restricted dynamic Steiner trees for scalable multicast in datagram networks. *IEEE/ACM Transactions on Networking*, 6(3):286–297, June 1998.
[12] Sriram Raghavan, G. Manimaran, and C. Siva Ram Murthy. A rearrangeable algorithm for the construction of delay-constraint dynamic multicast trees. *IEEE/ACM Transactions on Networking*, 7(4):514–529, August 1999.
[13] Tawfig Alrabiah and Taieb Znati. Delay-constrained, low-cost multicast routing in multimedia networks. *J. Parallel Distrib. Comput.*, 61(9):1307–1336, 2001.
[14] Mehrdad Parsa, Qing Zhu, and J. J. Garcia-Luna-Aceves. An iterative algorithm for delay-constrained minimum-cost multicasting. *IEEE/ACM Transactions on Networking*, 6(4):461–474, 1998.
[15] Bernard M. Waxman. Dynamic Steiner tree problem. *SIAM J. Discrete Math.*, 4:364–384, 1991.
[16] Leslie S. Liu and Roger Zimmermann. Active: A low latency p2p live streaming architecture. January 2005.
[17] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. BRITE: Universal topology generation from a user's perspective. Technical Report BUCS-TR2001-003, Boston University, January 2001.