Building Multicast Acknowledgment Trees

Christian Maihöfer, Kurt Rothermel

University of Stuttgart
Institute of Parallel and Distributed High-Performance Systems (IPVR)
Breitwiesenstr. 20-22
D-70565 Stuttgart

EMail: [Christian.Maihoefer|Kurt.Rothermel]@informatik.uni-stuttgart.de

Abstract

To avoid the well-known implosion problem, the majority of reliable multicast protocols use hierarchical structures to pass acknowledgment messages back to the sender. In most cases, a technique called expanding ring search (ERS) is used to construct the acknowledgment tree. In this paper we analyse ERS by simulation studies. ERS is a simple and fault tolerant approach, but our simulation results show that it has disadvantages like great reliance on the multicast routing protocol and poor scalability. If the background load exceeds a critical level, the message overhead rises exponentially. In this paper, we will present an alternative approach based on a so-called token repository service. The token repository service stores a token for each successor, a node in an ACK tree can accept. To find a node to connect to, the searching node just retrieves a token from the token repository. We describe how such a service can be implemented in a way that it can handle a large number of multicast groups. Our simulation results show that the token repository service is scalable, independent of the multicast routing protocol and builds well-shaped ACK trees, causing message delays that are comparable to ERS or even lower.

1 Introduction

Efficient one-to-many communication is a prerequisite for many new applications, e.g. news and software distribution, distributed computing and computer supported cooperative work. IP multicast [DC85, Dee89] is already available in the Internet but only with best effort semantics. Several papers have addressed the issue of developing a scalable reliable multicast protocol. All solutions are based on the idea that the successful delivery is controlled by some kind of ACK message returned from the receivers to the sender. The class of sender-based and receiver-based protocols [PTK94, LG96], where all control messages must be processed by the sender node can cause the well known implosion problem. A receiver of a multicast group sends all control messages (positive or negative acknowledgments) to the sender of the message. If the multicast group is large, i.e. has a huge amount of receivers, the network near to the sender and the sender itself becomes congested.

To guarantee scalability only tree-based approaches like TMTP [YGS95], RMTP [LP96], RMTP-II [WEA98] or LGMP [Hof96] are applicable. To avoid the implosion problem, all receivers are organized in a so-called acknowledgment tree (ACK tree). Instead of sending all control messages (positive or negative acknowledgments) to the sender, each receiver contacts only its direct predecessor in the tree. Leaf nodes send a positive acknowledgment to their predecessor to indicate that they have received a message correctly, respectively a negative acknowledgment if they have recognized an incorrect or lost message. The behavior of an inner node in the ACK tree strongly depends on the underlying transport protocol. In any case, an inner node is responsible for retransmitting lost messages to its successors. Inner nodes also send control messages to its predecessor when all of their successors in the ACK tree have received the message. When the root node eventually receives the aggregated acknowledgments from all direct successor nodes, the message has been reliably delivered to all receivers.

But how is such an ACK tree established? In most cases a technique called expanding ring search (ERS) is used to construct the ACK tree. In this paper we present simulations that examine the behavior of ERS. Our results show that ERS has a poor scalability, its performance heavily relies on the multicast routing protocol and that the generated ACK trees are not well-shaped. To overcome these deficiencies, we sketch a new approach to

build ACK trees. Our idea is to combine ERS with a so-called token repository service. The repository service stores tokens, which indicate at which ACK tree node a new multicast group member can join the tree. When a new member wants to join the tree, it simply asks the token repository service for a token of that group. Our approach provides fault tolerance, scalability due to decreased network traffic, and message delays that are comparable to ERS or even lower. Furthermore the performance of the token repository service is independent of the applied multicast routing protocol.

The remainder of this paper is organized as follows. In the next section, the principle of ERS and related work is discussed. Section 3 gives an overview of the proposed token repository service and describes the group management operations. In Section 4, we present simulation results for ERS and the token repository service. Finally, we conclude with a brief summary.

2 Background and Related Work

ERS is a common technique that was first used with broadcasting to search for resources in a network without the knowledge, where such a resource can be [Bog83]. With basic ERS the joining node looks for a predecessor node in the ACK tree by sending multicast search messages with increasing time-to-live (TTL). The first search message is sent with a TTL of one, i.e. the search message is limited to the LAN of the sender. If an on-tree node, i.e. a node that is already in the ACK tree, receives this request and is able to accept further successor nodes it replies with an acknowledgment. The searching node is then able to join the tree with the replying node as the predecessor. If no node answers within a certain amount of time, the TTL is increased by one and a new search message is sent. The joining node repeats this process until it receives an answer or the maximum TTL of 255 is reached. Increasing the TTL step by step is done in order to decrease the network load and to find a predecessor node that is close to the searching node. As a result, a continuously increasing region is covered (see Figure 1). This basic method is described by [YGS95]. We refer to this approach as off-tree initiated method because off-tree nodes, i.e. nodes that are not already in the ACK tree, are responsible for finding a predecessor node in the ACK tree.

There is one major variation to this off-tree initiated method that we will call the on-tree initiated method. With the on-tree initiated approach, nodes that are already in the ACK tree and that are able to accept further successors periodically send a multicast invitation message. To reduce network load and to ensure locality these invitation messages can also be sent with an increasing time-to-live value. [Hof96] uses such an on-tree initiated approach with a special TTL distribution that ensures that with increasing scope the frequency of invitation messages decreases. If a joining node receives more than one invitation message it can choose the best predecessor based on quality metrics [Hof96]. Some protocols like Lorax [LLG96] or TRAM [CEA98] combine on-tree and off-tree initiated approaches.

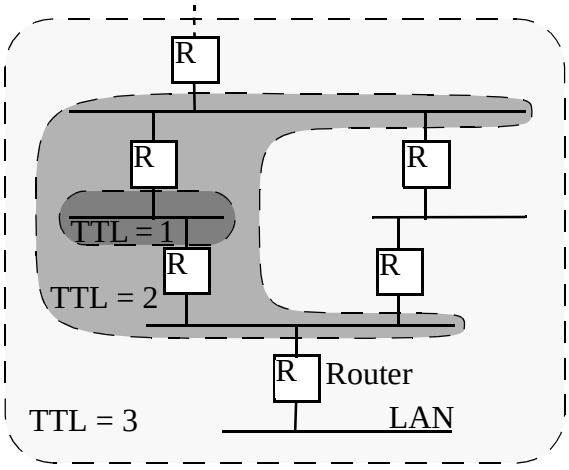

Figure 1: Increasing search radius of ERS

Our simulation studies consider only the off-tree initiated approach. One disadvantage of this approach compared to the on-tree initiated approach is that the underlying network must provide bidirectional multicast, this means all joining nodes must be able to send multicast messages. The advantage of this approach is that the network is not congested due to periodically sent multicast invitation messages. Note that these messages are sent even if no new node wants to join the tree.

To give a complete overview it must be mentioned that some protocols extend the routers' functionality [LC98, LGT98, PPV98]. With support on the network level it is possible that no separate ACK tree must be established. Instead the multicast routing tree must be used for both tasks, routing and reliable message delivery. However, this approach requires changes in routers, and even in some of these protocols the routers itself are respon-

sible for layer 4 error recovery, which does not conform with the layered architecture of the Internet.

3 The Token Repository Service Approach for Building ACK Trees

3.1 Overview of the Token Repository Service

In this section we describe a new method to build ACK trees. We assume that there exists a global token repository service, which is responsible for all multicast groups. When a *k*-bounded node, i.e. a node that is able to accept up to *k* successors, joins a group, *k* tokens are created and stored in the repository, where the joining node is called the tokens' owner. Each of these tokens contains at least the information about their multicast group and owner. Besides this necessary information, a token may contain further information, such as the height of the owner in the ACK tree, which is used to create well-shaped ACK trees. If a node wants to join the tree, it simply asks the token repository service for a token. If there is more than one token available, the "best" token with regard to the height is chosen.

The token repository depicted in Figure 2 contains tokens due to former join group operations. Node *N* wants to join the ACK tree and therefore asks the repository for a token. The token repository service delivers one token to *N* and removes it out of the repository. The token gives the right to connect to node N_2 in the ACK tree. If *N* wants to accept *k* successor nodes itself, *k* tokens of *N* are created in the repository service.

Due to scalability reasons, token information is distributed in a hierarchical manner. In other words, the token repository service is implemented by a hierarchy of servers. Each token repository server, repServer in short, is responsible for a particular domain, consisting of a set of nodes (see Figure 3 and Figure 4). The domains of leaf repServers consists of distinct nodes. The domain of an inner repServer is the union of its successors' domains. Typically all nodes within a domain are closer to each other than to nodes in another domain, where closer can mean lower delay, less hops or lower message loss probability, for example.

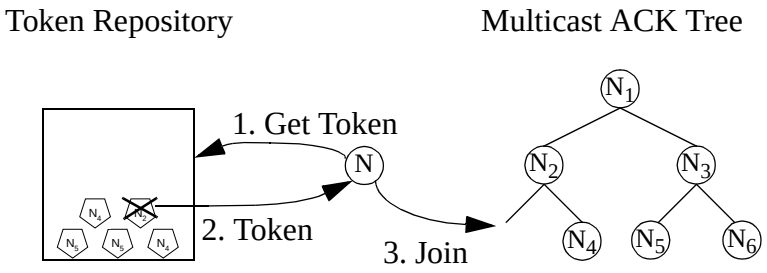

Figure 2: Token repository service

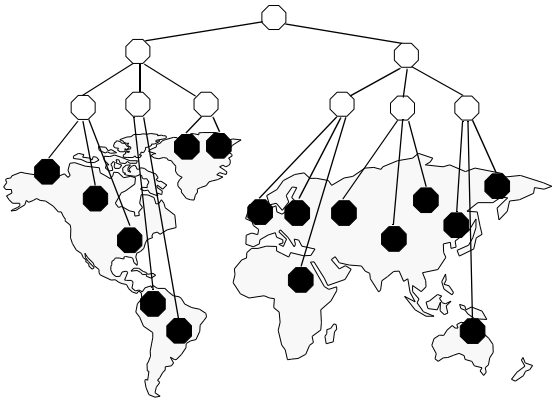

Figure 3: Hierarchical repServer structure

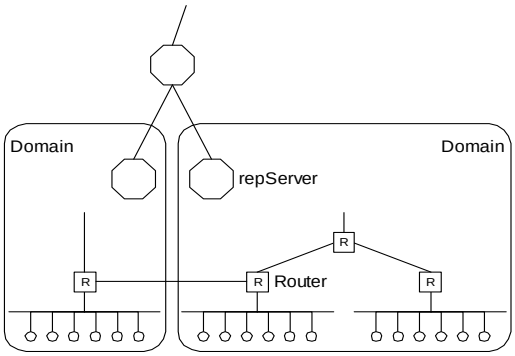

Figure 4: Domain structure

Tokens are stored at leaf repServers only. Each leaf repServer provides the group management service, i.e. create group, join group, leave group and delete group operations, to all clients in its domain. The inner servers are necessary to process a token search if no token is available locally. Inner servers keep track for which groups tokens are available in each of its subhierarchies. To search for a token of a particular group, the predecessor server in the repServer hierarchy must be asked recursively until one is found that has a token of this group in its subhierarchy. In the second phase of the search, the search request is forwarded along the path from the inner server to the leaf server that stores tokens of this group, and finally the token is handed over to the searching node. Due to this hierarchical structure, it is possible to model locality and to ensure that joining nodes get near predecessor nodes in the ACK tree. In the following section we will explain the group management operations in more detail.

3.2 Group Management Operations

For creating and maintaining multicast groups, the four operations to create a group, delete a group, join a group and leave a group are necessary. As already mentioned above, all operations are issued at leaf repServers.

When a new group is created, the leaf repServer must create tokens and establish the search structure in the repServer hierarchy. The number of tokens to be created depends on the processing power and reliability of the ACK tree node, the more reliable and more powerful a node is the more tokens are created. While the created tokens are stored locally on the leaf repServer, the establishment of the search structure involves all repServers on the path from the leaf repServer to the root repServer. The leaf repServer forwards the create group operation to its predecessor, which itself forwards it to its predecessor. This continues until the root repServer is reached. Every involved repServer creates a search structure to allow subsequent join operations to search for tokens.

The search structure associated with a group is a bit vector, consisting of k entries, where k is the number of successor repServers. Each bit in the vector represents one successor repServer where the bit is set to one if there is a token of this group in the successor's domain.

When the search structures are created due to a create group operation, each search structure contains exactly one bit set to one, corresponding to the successor repServer that forwards the create group operation. All other bits in the vector are set to zero. Since the token repository service is responsible for all existing reliable multicast groups, each inner server has in general several search structures, one for each known group in this domain.

When a new node wants to join the ACK tree, it asks its leaf repServer for delivering a token. If the requested token is locally available, the leaf repServer returns this token to the joining node, deletes this token and creates a new set of tokens for the joining node. Otherwise, the tree structure of the token repository service is used to search for a token. The token search is forwarded in a leaf-to-root-direction until a repServer is found that has a search structure for this group with at least one bit set to one. Then the second search phase in root-to-leaf-direction is started. The search request is forwarded according to the information in the search structures until a leaf repServer is reached. This leaf repServer hands over a token to the searching leaf repServer, which itself hands over the token to

the joining node and creates new tokens for this node. In general, if a token must be delivered either to a new multicast group member or a searching leaf node and there is more than one token for this group available, always the “best” token is taken. The best token can be the token which leads to the least height in the ACK tree or the least error probability for example.

A join operation may require the search structures to be updated. On the one hand, it is possible that the token delivering leaf repServer hands over its last token and therefore cannot deliver further tokens. On the other hand, the searching leaf repServer now has tokens itself and hence is able to deliver tokens to other leaf repServers. In both cases, an update operation is sent to the predecessor indicating that the search structure is to be updated accordingly. If this affects the token availability in the domain of the predecessor node, the update operation if forwarded again to the next predecessor.

When a node leaves a group, its tokens at the leaf repServer are deleted and if necessary, an update operation to the predecessor initiated. Furthermore, a new token must be created for the leaving node's owner at the corresponding leaf repServer and again if necessary, an update operation to the predecessor repServer initiated.

A group is deleted in the token repository service by removing all tokens and search structures of this group. This is achieved by forwarding the delete group operation to the predecessor until the root node is reached and to every successor repServer that has a token for this group in its domain.

3.3 Fault Tolerance

In the previous section, we have described the token repository service approach without taking into account possible error conditions like repServer crashes or network partitioning. Our protocol is able to handle such errors as we will see in this section. We do not consider message loss here since we assume a reliable unicast transport protocol like TCP.

Due to performance reasons tokens and search structures are stored in volatile memory. The only information stored in stable memory is the tree structure of the token repository service, i.e. every repServer stores its predecessor and successor servers.

If a repServer fails, its search structures and tokens are lost and will not be explicitly recovered after restarting. To cope with this information loss we assume, that the update op-

erations are sent periodically which leads to a recovering of the search structures. If a leaf repServer has lost its token and cannot process a join operation, a token search is started. If the search does not succeed, i.e. no token is found at all, ERS is used. After a predecessor node in the ACK tree is found with ERS, the new created tokens are handed over to the token repository service. Consequently, subsequent join operations can be handled by the token repository service again. The effects of a network partition are similar to a repServer failure. A partition may lead to inconsistent search structure and token information but at least with ERS it is always possible to join a group and to deliver new tokens to the repository service.

Due to these mechanisms the token repository service is able to tolerate any number of node failures and network partitions. Even if no repServer at all is available, it is always ensured that a new node can join the ACK tree.

4 Simulations

We have run simulations with the NS2 Network Simulator [NS2] to compare ERS with the token repository service with regards to the quality of the created ACK tree, scalability and round trip delay between leaf nodes and the ACK tree root node.

4.1 Simulation Scenario

Our scenarios consist of different networks with 100 to almost 2000 nodes connected by duplex links. The networks are generated with Tiers [Tiers]. The bandwidths are in general 10Mbps for LANs, 100Mbps for MANs and 1000Mbps for WANs. The link delays are chosen randomly from 1ms to 3ms for LANs, 1ms to 8ms for MANs and from 5ms to 19ms for WANs. Although all results presented in this paper are based on networks generated with Tiers we have also used GT-ITM [GT-ITM] to generate flat, stub and hierarchical networks. The results between this two network generators differ only slightly.

We use the distance vector multicast routing protocol (DVMRP) [WPD88] and core based tree (CBT) [Bal97]. All routers in the network use drop tail queues. Our simulation scenario does not consider error situations, i.e. messages are delivered reliably, nodes do not fail and the network is not partitioned.

The used token repository service is a complete two bounded tree, i.e. every inner repServer in the tree has two successor repServers. It consists of 8 leaf repServers which results in a total amount of 15 repServers for the whole tree. For minimizing the round trip delay in the created ACK tree it is crucial to form domains consisting of nodes with small delays between them. Therefore, the simulations to measure the round trip delay are run with a repository service tree configured by hand, while all other simulations use a non optimized repository service tree.

We have run simulations with different background traffic conditions. The background traffic consists of randomly placed senders and receivers of TCP traffic. The traffic of a sender is exponentially distributed. Although our simulations were run on a Sun Ultra 2 with 296MHz and 1,4GB main memory it was unfortunately not possible to simulate highly background load with the given network bandwidth. Therefore, we had to decrease the bandwidths to 100Kbps for LANs, 1Mbps for MANs and 10Mbps for WANs to run the simulations with background traffic.

In the simulations, between 50 and 200 randomly chosen nodes join the ACK tree. All presented results are based on join operations that are evenly distributed in time and space. We have also run further simulations with exponential and normal distributions, which leads to the same results.

4.2 Simulation Results

In the first experiment, we have examined if the ACK trees built are well-shaped. We assume that the ACK tree is *k*-bounded by a constant *k* that is the same for all nodes in the tree. This means every on-tree node can accept up to *k* successors. We say an ACK tree is well-shaped if its height is almost the minimum height of a *k*-bounded tree with the same amount of nodes. Well-shaped ACK trees are advantageous because in general the round trip delay between the root node and the leaf nodes is lower due to less intermediate nodes. Also it is more critical if inner nodes fail or leave the tree than if leaf nodes do. This is another advantage of well-shaped ACK trees because they have less inner nodes and more leaf nodes and thus the probability that the tree must be reorganized is lower.

Our first result is that ERS does not produce well-shaped ACK trees. While varying *k* in the range from 2 to 40, the average branching varies only between 2 and 3 with DVMRP

routing and between 2 and 10 with CBT routing. In a simulation with 50 join operations in a 251 nodes network the height of the trees is about 62% to 200% greater with DVMRP and 100% to 325% with CBT than the minimum height of a *k*-bounded tree. The token repository service produces better shaped ACK trees. The average branching varies between 2 and 7 and the height of the trees is only about 33% to 100% greater than the optimum.

In Figure 5, we present the amount of messages sent to build the ACK tree for the two routing protocols depending on different number of nodes in the network. Note that the messages sent by ERS are multicast messages, while the token repository service sends unicast messages.

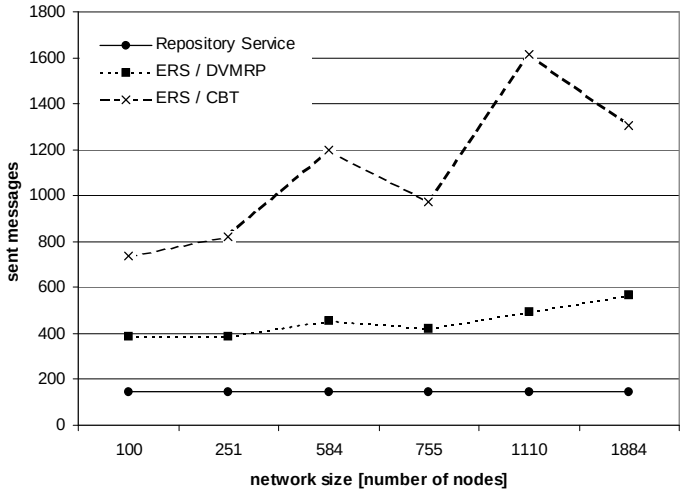

Figure 5: Messages sent depending on the network size

The token repository service sends the smallest number of messages and is independent of the network size. ERS scales not very well since more messages must be sent and the amount depends heavily on the network size. ERS in combination with CBT routing produces the worst results. This behavior can be explained with the message dissemination starting always from the same core node in the network. The ACK tree nodes that are near this core node will soon be satisfied with successor nodes and therefore the ERS search must cover a large network area to discover ACK tree nodes that are able to accept further successor nodes. The bends in the curve are also the effects of the core node since the amount of messages depends heavily on the place, where the core node is located in the network and the distribution of the join operations. Our result shows that only the token repository service scales well with the network size.

The results in Figure 6 show that the amount of messages received from on-tree nodes depends heavily on the timeout parameter of the ERS search algorithm and the background load of the network. The simulation scenario consists of a network with 251 nodes and DVMRP routing protocol.

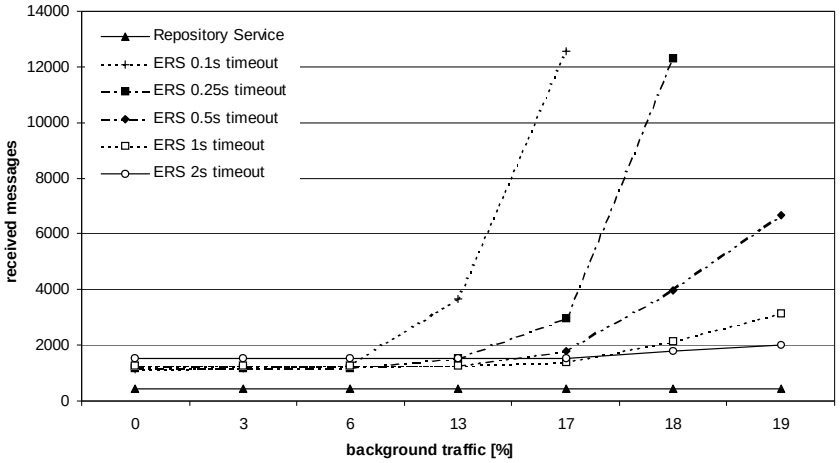

Figure 6: Messages received depending on the background traffic

The background load is measured as the average link load in the whole network. 10% background load can mean that each link has a load of 10% of its maximum or that 10% of the links are at their maximum load level and the remaining 90% have no load at all.

The timeout parameter determines after which time a new search with an increased TTL is started. We vary the timeout between 0.1 and 2s per TTL hop. This means if the timeout is set to 1s and the multicast search message has a TTL of 10 then the searching node waits 10s until a new search is started.

If ERS is used and the background load is high, the amount of received messages rises exponentially. This effect is the worse the smaller the timeout parameter is set. On the other hand, increasing the timeout parameter is only practical in a small range because this would increase the time necessary to join the ACK tree. Furthermore, it could be seen in the chart, that with an increased timeout more messages must be received in the case of low background traffic. The increased time necessary to join the ACK tree is the reason for this effect. If it takes longer to join the ACK tree, it also takes longer before the joining node itself is able to accept successor nodes and therefore other joining nodes must possibly search in a larger scope to find a predecessor node. The use of the token repository service results in the least number of messages that must be processed by the network and

ACK tree nodes. Our result shows that in contrast to ERS, the token repository service scales very well with background traffic.

We have compared the ACK tree round trip delay between ERS and the token repository service. The round trip delay is the time period between sending a multicast message and receiving the last aggregated acknowledgment at the ACK tree root node. Our simulation results show that the token repository service leads to only slightly increased round trip times compared to ERS with DVMRP. In our simulation runs with different background loads and different randomly chosen join operations the difference is only about 6%. If ERS with CBT routing is used, the token repository service achieves even better results than ERS. The round trip delays with the token repository service are about 10% better than with ERS.

Our results indicate that ERS depends greatly on the multicast routing protocol, the network size and the background traffic. The scalability of ERS in networks with higher background load is very poor. Furthermore, we have shown that ERS leads to a higher network load and processing power requirements of the ACK tree nodes the more nodes the network contain. Although ERS always searches for the closest predecessor our simulations show that the token repository service leads to round trip delays in the ACK tree that are almost identical to ERS or even lower if a core based multicast routing mechanism is used.

5 Conclusions

We have presented a new fault tolerant and efficient approach for building multicast ACK trees called token repository service. The token repository service has been simulated and its behavior compared to expanding ring search in different situations and with different parameters. It has been shown that the token repository service scales better than expanding ring search and that it results in better shaped ACK trees.

We are currently working on improvements for the token repository service. The metrics we consider so far when choosing a predecessor node is the height of the ACK tree, the distance from the joining node to the predecessor node and the processing power of the predecessor. It is possible to extend the token repository service with further metrics such as the error probability of the predecessor node. To react on changing conditions it may

be reasonable to allow dynamically changes in these values. For example, the processing power is modeled by the amount of created tokens. If the processing power changes, it is possible to add more tokens or to delete some tokens if not all of them are already used. If we use the error probability as a further metric, the ACK tree node can report changes of its error probability, measured by the amount of lost packets for example, to its leaf repServer, too. With these mechanisms we are confident that the quality of the created ACK trees can be further improved.

6 References

[Bal97] Ballardie, A.: Core Based Trees (CBT version 2) Multicast Routing, RFC 2189, 1997

[Bog83] Boggs, D.: Internet Broadcasting, Ph.D. Th., XEROX Palo Alto Research Center, Technical Report CSL-83-3, 1983

[CEA98] Chiu, D. M.; Hurst, S.; Kadansky, J.; Wesley, J.: TRAM: A Tree-based Reliable Multicast Protocol, Sun Microsystems Laboratories Technical Report Series, TR-98-66, 1998

[DC85] Deering, S.; Cheriton, D.: Host Groups: A Multicast Extension to the Internet Protocol, RFC 966, 1985

[Dee89] Deering, S.: Host extensions for IP multicasting, RFC 1112, 1989

[GT-ITM] GT-ITM: Georgia Tech Internetwork Topology Models, http://www.cc.gatech.edu/fac/Ellen.Zegura/graphs.html

[Hof96] Hofmann, M.: Adding Scalability to Transport Level Multicast, Lecture Notes in Computer Science, No. 1185, 1996, pages 41-55

[LGT98] Lehman, L. W., Garland, S., Tennenhouse, D. L.: Active Reliable Multicast, Proceedings of the Conference on Computer Communications (IEEE Infocom), 1998, page 581

[LG96] Levine, B.N.; Garcia-Luna-Aceves, J.J.: A comparison of known classes of reliable multicast protocols, Proceedings of the IEEE International Conference on Network Protocols, 1996, pages 112-121

[LLG96] Levine, B.N.; Lavo, D.B.; Garcia-Luna-Aceves, J.J.: The case for reliable concurrent multicasting using shared ACK trees, Proceedings of the fourth ACM International Conference on Multimedia, 1996, pages 365-376

[LC98] Li, D.; Cheriton D. R.: OTERS (On-Tree Efficient Recovery using Subcasting): A Reliable Multicast Protocol, Proceedings of 6th IEEE International Conference on Network Protocols (ICNP'98), 1998, pages 237-245

[LP96] Lin, J.C.; Paul, S.: RMTP: A Reliable Multicast Transport Protocol, Proceedings of the Conference on Computer Communications (IEEE Infocom), 1996, pages 1414-1424

[NS2] UCB/LBNL/VINT Network Simulator - ns (version 2), http://www-mash.cs.berkeley.edu/ns/ns.html

[PPV98] Papadopoulos, C., Parulkar, G., Varghese, G.: An error control scheme for large-scale multicast applications, Proceedings of the Conference on Computer Communications (IEEE Infocom), 1998, page 1188

[PTK94] Pingali, S.; Towsley, D.; Kurose, F.: A Comparison of Sender-Initiated and Receiver-Initiated Reliable Multicast Protocols, Proceedings of ACM SIGMETRICS, 1994, pages 221-230

[Tiers] Tiers Topology Generator, http://www.geocities.com/ResearchTriangle/3867/sourcecode.html

[WPD88] Waitzman, D., Partridge, C., Deering, S.E.: Distance Vector Multicast Routing Protocol, RFC 1075, 1988

[WEA98] Whetten, B.; Basavaiah, M.; Paul, S.; Montgomery, T.; Rastogi, N.; Conlan, J.; Yeh, T.: The RMTP-II Protocol, Internet Draft, work in progress, 1998

[YGS95] Yavatkar, R.; Griffioen, J.; Sudan, M.: A reliable dissemination protocol for interactive collaborative applications, Proceedings of the third ACM International Conference on Multimedia, 1995, pages 333-344