Optimal Placement of NAK-Suppressing Agents for Reliable Multicast: A Partial Deployment Case*

[Extended Abstract]

Ovidiu Daescu Raja Jothi Balaji Raghavachari Kamil Sarac

Department of Computer Science
University of Texas at Dallas
Richardson, TX 75083, USA
[daescu, raja, rbk, ksarac]@utdallas.edu

ABSTRACT

In reliable multicast, receivers use negative acknowledgments (NAKs) to inform the sender about their packet loss. The growth in the number of NAK messages received by the sender results in the well-known feedback (or NAK) implosion problem. Therefore, one important issue for reliable multicast protocols is to utilize an effective mechanism to collect NAK messages from the receivers. One way to avoid feedback implosion at the sender site is to place NAK-suppression agents on the internal nodes (routers) of the network. These agents will forward a single copy of the incoming NAK messages toward the sender site and will suppress additional redundant copies coming from the receivers.

In this paper, we consider an agent placement (activation) problem for reliable multicast. First, we assume a network environment where a number of internal nodes (routers) have NAK-suppression capabilities. Then, we try to select a subset of these nodes for NAK suppression task for a given reliable multicast application. Our main selection criteria is to choose a minimum number of such nodes for the task and have a notion of load-balancing among them. In this context, we study two agent activation problems: the *Load-Balanced Agent Activation Problem* (LBAAP) and the *Budgeted Agent Activation Problem* (BAAP) and present efficient algorithms for optimal activation of agents. The problems that we consider in this paper are generalized versions of the respective problems introduced by Daescu et al. [3].

Categories and Subject Descriptors

G.1.6 [**Numerical Analysis**]: Optimization—*constrained optimization*; G.2.1 [**Discrete Mathematics**]: Combinatorics—*combinatorial algorithms*

*Research supported by NSF under grant CCR-9820902.

SAC'04, March 14-17, 2004, Nicosia, Cyprus

General Terms

Algorithms, Design

Keywords

Reliable multicast, feedback implosion, NAK-suppression, system design, graph theory

1. INTRODUCTION

A number of multicast applications require timely receiver feedback. In a reliable multicast application, receivers use negative acknowledgments (NAKs) to inform the source about packet loss so that the source can retransmit the missing data. However, as the size of the receiver population increases, collecting receiver feedback becomes a significant problem for the source. This problem is known as the multicast feedback (or NAK) implosion problem[13, 16].

Bombardment of the sender by numerous, often redundant, feedback messages from receivers results in feedback implosion problem. Typically, loss of a packet triggers one or more receivers, which did not receive that packet, to send NAK messages to the sender. For a packet loss, the sender could potentially receive multiple NAKs. After all, it is enough that the sender receives just one NAK for a packet loss rather than receiving redundant NAKs. Processing redundant NAKs could easily impede the performance of the sender, and thus the network as a whole.

One common approach to prevent feedback implosion in reliable multicast is to use NAK-suppressing agents in the network. These agents are software modules located in routers (or in server machines co-located with routers). Each NAK-suppressing agent incurs processing and memory overheads for the router hosting it. This overhead depends on the number of reliable multicast groups for which the node is serving as a NAK-suppression agent. Considering the processing overhead incurred on the routers, in this paper, we assume that only a subset of the routers (i.e. high end more powerful routers) have the agent functionality in the network. This is in contrast to [3], in which all the routers have the agent functionality.

Given a large number of reliable multicast groups in a network, one important task is to make an intelligent selection of agent nodes to be activated for a given reliable multicast group. Activating all the agent nodes is an obvious solution,

but the overheads, at each agent router, due to additional processing could slow down the system. Moreover, nodes with high degree could become potential hot spots when compared to nodes with relatively low degree. One important selection criteria that we use in this paper is to consider a notion of load balancing in identifying the agent locations among a number of alternatives. Without such consideration, the performance of the routers having excessive agent load may degrade and these routers may become potential traffic hot spots.

1.1 Problem Formulation

Throughout this paper, by "agent placement" we actually mean agent activation. To prevent redundant NAKs reaching the sender, we propose placing (activating) NAK-suppressing agents at certain nodes so that all but one NAK gets suppressed. This certainly reduces the number of NAKs received by the sender. The functionality of a NAK-suppressing agent is that of suppressing redundant NAKs, thus allowing only one NAK to pass through.

We define the "load" on an agent to be the number of NAKs it suppresses. Ideally, we would want to activate agents at nodes such that the load on all agents are almost the same. A node, whose agent is agent is activated, is overloaded, if it suppresses more than, say, L NAKs. The load at an overloaded node can be reduced by activating agents at nodes below that node in the multicast tree, thereby directly reducing the number of NAKs passing through that node.

In this paper, we consider the efficient way to activate agents on multicast trees, where the leaf nodes of a tree are the multicast group receivers and the internal nodes are the routers. In general, not all the routers in the network have same capabilities. Some routers are very powerful, capable of making major decisions during catastrophic failures, while others just route traffic. Thus, we classify routers into two kinds: (i) active (special) routers and (ii) passive routers. Active routers are the ones that have high processing capabilities, huge buffers and important decision-making algorithms. Passive are the ones with simple configurations whose sole purpose is to transfer (route) the traffic. Since agent functionality typically require reasonable buffer space and processing capabilities, we restrict the placement of agent modules to active routers only, i.e., agents are available for activation only on active routers. This is in contrast to the work by Daescu et al. [3], in which agents modules are available on all routers. Unlike [3], we consider a more realistic model in which agents are available only on a subset of routers.

In what follows, we present the formal definition of the two agent activation problems discussed in this paper: (i) the load-balanced agent activation problem (LBAAP) and (ii) the budgeted agent activation problem (BAAP).

LBAAP: Given a multicast tree T rooted at node r, where r is the multicast source node (sender). The leaves of T are the multicast group receivers and the internal nodes are the routers. Under this setting, given a load bound L, the load-balanced agent activation problem asks for the activation of minimum number of agents at active routers such that no agent handles a load more than L.

BAAP: Given a multicast tree T rooted at node r, where r is the the multicast source node (sender). The leaves of T are the multicast group receivers and the internal nodes are the routers. Given K agents, the budgeted agent activation problem asks for the activation of K agents at active routers such that the maximum load L on an agent is minimized. Note that, minimizing the maximum load on an agent is same as that of balancing the load among the K agents.

Throughout this paper, we assume that the multicast tree topologies are known beforehand, at least at the intra-domain level. Several tools for discovering multicast tree topologies that span multiple domains in the Internet have been proposed in recent years (see e.g., [4]). Normally, network providers (ISPs) have enough information about the multicast tree topologies within their domain. If the multicast tree spans several domains, our algorithms can be run locally at the intra-domain level for the portion of the multicast tree that spans that particular domain. In this way, for trees spanning multiple domains, our algorithm can be used at each level.

1.2 Related Work

Load balanced agent location (or placement/activation) problem resembles two well-known graph theoretic problems: the k-median problem and the facility location problem. Given a connected undirected graph with n nodes, the k-median problem is to select k nodes as service centers that will minimize the sum of the costs (i.e. distances) of all other nodes (customers) to their respective nearest service center among the k selected service centers. In [17], Tamir studies the k-median problem on trees and gives an optimal algorithm with a running time of $O(kn^2)$. Li et al. [11] use a similar approach to optimally place web proxies on a tree network architecture with a web server at the root of the tree. Their main objective is to minimize the overall latency in serving client requests from the leaves of the tree. In [14], Qiu et al. study the same problem on a graph topology and propose various heuristics. In [10], Krishnan et al. study the problem of optimal placement of network (web) caches. Their goal is to minimize the overall flow or the average delay by placing a given number of caches in the network. In [15], Shah et al. deal with the k-median problem in the context of content-based multicast. They define a filter placement problem as a version of k-median problem and provide two algorithms for optimal filter placement with the objectives of minimizing mean total network bandwidth utilization and mean information delivery delay. In addition, a significant amount of study has been done on the placement of agents in the context of reliable multicast [5, 8, 12]. The main objective of these works is to reduce the number of retransmissions, latency and resource utilization.

The concept of placing k agents on a graph with the goal of load balancing has been addressed in the context of facility location problems [7]. Facility location problem on graphs is similar to k-median problem, except that there is an additional cost associated in opening a facility node. In [6], Guha et al. introduced the Load Balanced Facility Location problem on graphs wherein the constraint of having a minimum load on facility nodes is placed on the original definition of the problem. They prove that this version of the problem is NP-Complete and present a constant factor approximation algorithm for it. In [9], Jothi and Raghavachari study the placement of repair servers in a network for supporting reliable multicast application. However, their goal is to statically identify a number of nodes/routers in the network such that they will be used by most of the multi-

cast connections in the network. In contrast, in our work, we consider the multicast tree topologies of the currently existing reliable multicast sessions and activate a subset of agents in the network dynamically.

In addition to reliable multicast, there are other multicast applications that use agent support from within the network. In [15], Shah et al. use active filtering services in the network to improve bandwidth usage efficiency in the context of content based multicast; and in [1, 2], the authors use active network agents to provide support for a group of multicast services including feedback aggregation and suppression, subcasting, and directed multicast.

1.3 Our Contributions

In this paper, we study two NAK-suppressing agent activation problems: (i) the LBAAP and (ii) the BAAP. Both these problems are generalized versions of the respective problems introduced by Daescu et al. [3]. In [3], all nodes have agent functionality. In our work, we consider the realistic model, in which agents functionality is available only on a subset of routers. In other words, problems considered in [3] are the special cases of the respective problems studied in this paper. For both LBAAP and BAAP, we present efficient algorithms for the optimal activation of agents on a multicast tree.

2. AGENT ACTIVATION PROBLEM ON A SINGLE MULTICAST TREE

In this section we present efficient algorithms for the agent activation problem on a single multicast tree. A typical application scenario for this case is as follows. An ISP offers a number of value-added network services including NAK suppression for reliable multicast application, to his/her customers. Based on the type(s) of service requested by the customer, the ISP usually charges a fixed fee based on the number of services requested or a variable fee based on the quality of service among others. The first scenario corresponds to that of LBAAP, where the ISP charges a fixed fee to provide NAK suppression service to its reliable multicast customers. Under this fixed fee model, ISPs would naturally want to provide the service in an efficient (cost-saving) way. This directly translates to smart selection of agent locations such that the number of agents used to provide the desired service is as small as possible. The fewer the number of agents, the cheaper and better it is for the ISPs.

The second scenario where the ISP charges the multicast service user based on the number of agents corresponds to the BAAP problem. Under this model, the service user informs the number of agents that he/she wants for his/her multicast service. This usually depends on the amount of money that the user can afford to spend. Under the budget constraints, the user typically wants the best service possible, i.e. most effective NAK suppression.

When an ISP receives a request for reliable multicast agent support task, it will dynamically discover the multicast tree topology in the network and then run one of the above algorithms to identify the agent nodes at which the agent functionality needs to be activated to serve the request. In order to accommodate the changes in the underlying multicast tree topology, the ISP will periodically collect the tree topology and invoke the algorithm to update the agents. As said earlier, several tools for discovering multicast tree topologies that span multiple domains in the Internet have been proposed in recent years (see e.g., [4]). If the multicast tree spans several domains, our algorithms can be run locally at the intra-domain level for the portion of the multicast tree that spans that particular domain. In this way, for trees spanning multiple domains, our algorithm can be used at each level.

2.1 The LBAAP

First we consider the Load Balanced Agent Activation Problem (LBAAP) on mulitcast trees. The input is a multicast tree T rooted at r. Root r of T represents the source or the sender of the multicast tree. The leaves of T are the multicast group receivers and internal nodes are the routers. Routers are classified into active and passive routers. Unlike in [3], agents modules are available only only at active routers. Each leaf node v_i has a weight of $w(v_i)$ that represents the number of receivers located at that leaf node (1 if the node is a single system, and a higher value if it is a gateway to a network). Let T_u be the subtree rooted at node u in T. The weight of an internal node u, $w(u)$, is defined to be the sum of the weights of all leaves in T_u. If an agent at an internal node u is activated, then its load is denoted by $ld(u)$. The load at an internal node, whose agent is activated, is the maximum number of NAK messages processed by that node in the event of a message loss or a failure. Since node u, whose agent is activated, replaces NAK messages generated within T_u by a single NAK message, activating an agent at u has the effect of replacing T_u by a single node whose weight is 1. Since the root of the tree could potentially receive many such NAKs, we safely assume that an agent at the root (or the active node connected to the root)is activated. Ideally, we would like the loads on the nodes, whose agents agents are activated, to be as balanced as possible.

In LBAAP, we are given a load constraint L, where L denotes the maximum load that an agent can handle, and we are asked to find the minimum number of agents at active routers that needs to be activated such that each node, whose agent is activated, has a load of at most L. We show that this problem is efficiently solvable by designing an algorithm that finds an optimal activation of the agents in quadratic time. We define "degree" of an internal node to be the number of children it has. In the calculation of degree, a child that is an internal node counts as 1, while a child that is a leaf v_i counts as $w(v_i)$, its weight. We can safely assume that the degree of each internal node is at most L. If the degree of each internal node exceeds L, the problem is infeasible for the given load constraint L, i.e., even activating the agent at every node will not provide a feasible solution.

Recall that agents are available only on active nodes (routers). Consider an optimal activation of agents with maximum load L. In an optimal solution, let u be an active node whose agent is activated, and let q be u's immediate ancestor whose agent is also activated (see Fig. 1). Then, de-activating the agent at u will cause the load at q to exceed L. Otherwise, the solution we started with is not optimal. We call q as the immediate active ancestor of u, and all active nodes that are descendants of q as the "cousins" of u. In Fig. 1, p, s and t are cousins of u.

We first show a canonical activation of agents, and then show how to find such a solution in quadratic time. First,

we move the agents as high as possible without violating the load constraint. In other words, if an agent at node u is activated, then it must be the case that if we de-activate the agent at u and activate the agent at u's immediate ancestor node p, which is an active node, then the load at that agent must exceed L. Otherwise, we can perform such a transformation and obtain another solution which uses the same number of agents. Second, we de-activate agents at active nodes of smaller weight and activate the agents at their respective active cousin nodes of higher weight (but at most L). In other words, if we activate the agent at an active node u, but not the agent at one of its cousin s, then $w(u) \geq w(s)$. We now show that there is always an optimal activation of agents that satisfies the above conditions.

LEMMA 1. *Let T be a multicast tree with root r, and let L be the maximum load for any agent on T. Consider a minimum number of agents that need to be activated on T to keep the maximum load to be less than or equal to L. Then, there is an optimal solution such that the following two conditions are satisfied:*

1. *If the agent at node u is de-activated and the agent at u's immediate active ancestor p is activated, then the load on the agent at p is more than L.*
2. *If an agent at active node u is activated, but not the agent at one of its cousins s, then $w(u) \geq w(s)$.*

PROOF. By "moving an agent up in the tree," we mean de-activating an agent at node u and activating the agent at u's immediate active ancestor node. We show that a given solution can be modified to satisfy the above two conditions without exceeding load L. Move an agent up in the tree if the resulting configuration is also feasible, i.e., each agent has a maximum load of L. Repeat this step until no agent can be moved up any further. This configuration of the agents satisfies condition 1 of the lemma. We now move an agent from an active node s to its active cousin node u if $w(s) < w(u) \leq L$. It is easy to verify that such a move generates another feasible solution, since the load on other agents either stays the same or decreases. In addition, condition 1 continues to be true after the move. Again, we repeat this step until condition 2 is satisfied. □

As an example consider the multicast tree in Figure 1. In this figure, node p has three descendants including t, s, and u with weights $w(s), w(t) < w(u) < L$. On the other hand, $w(p) = w(s)+w(t)+w(u)+w(x) > L$. Therefore, according to the above lemma, we activate the agent at active node u which is the child of p with the largest weight within the load bound L.

Having shown that there is a canonical optimal activation of the agents, we now show how to find one in quadratic time. We process the tree bottom-up and find a deepest node p whose weight is more than L. We activate the agent at an active node u, a descendant of p whose weight is more than any other active descendant of p, i.e., an internal active node of maximum weight among all such nodes that are descendants of p. Set the weight at node u to 1, which has the effect of replacing T_u by a single node with a weight of 1. The above step is repeated until the remaining tree has load at most L, which is handled by the agent at tree's root. The correctness of the algorithm follows from Lemma 1.

The following pseudo-code shows the algorithm. A global variable `nagents` counts the number of agents activated by the algorithm. This variable is initialized to 1, corresponding to the agent activated at the root. At the end of the algorithm, we set `ld(r) := w(r)`. A node u at which an agent is placed handles a load of `ld(u)`. For other nodes, the algorithm sets the variable to 0.

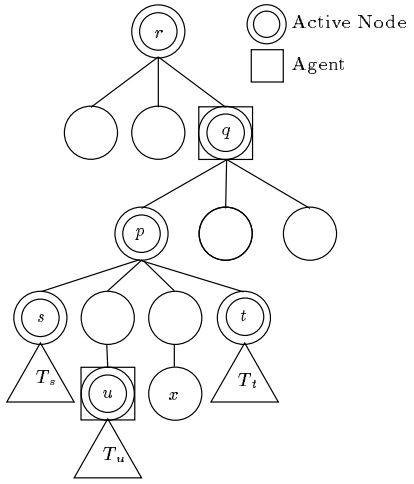

Figure 1: A sample scenario for LBAAP with $w(s), w(t), w(x) < w(u) < L$ and $w(p) > L$

```
LBAAP(T, r, L)
   nagents := 1
   ld(r) := Postorder(r)
   return nagents

Postorder(p) /* process subtree rooted at p */
   if p is a leaf node, return w(p)
   else
      load := 0
      for each child u of p do
         load := load + Postorder(u)
      while load > L do
         find an active descendant node u of p with
           maximum weight
         Activate the agent at u
         nagents := nagents + 1
         ld(u) := w(u)
         load := load - w(u) + 1
         w(u) := 1
      ld(p) := 0
      w(p) := load
      return w(p)
```

Processing each node takes linear time, since finding an active descendant node u of p with maximum weight takes linear time. Since there is a total of n nodes, the running time is $O(n^2)$.

THEOREM 1. *Algorithm LBAAP finds an optimal activation of agents with maximum load L for a given multicast tree T.*

PROOF. The proof is by induction on the number of agents activated by the algorithm. If the total load of the tree is less than or equal to L, then the algorithm activates the agent at r, which is clearly optimal. Otherwise, let u be the first node whose agent is activated by the algorithm when processing `Postorder(`p`)`, where u is an active descendant of u. By Lemma 1, there is an optimal solution that activates

the agent at node u, since load of p is more than L, and u is an active descendant of p with maximum load. Then, we can replace T_u by a single node with a weight of 1. By induction, in this smaller tree, our algorithm finds an optimal activation of agents. □

2.2 Budgeted Agent Activation Problem

We now consider the complementary problem, the budgeted agent activation problem (BAAP). Given a multicast tree T and K agents, find K nodes whose agents needs to activated to achieve load balancing. Load balancing in this context is defined as minimizing the maximum load L on a router. We show that this problem can be solved using the LBAAP algorithm as a subroutine with the parametric search technique. Suppose we "guess" the value of L. We solve LBAAP with this value of L and find the minimum number of agents N_L needed to keep the load of any router under L. If $N_L > K$, then our guess of L is too small, and we have to increase it. It is easy to show that N_L is a monotonic function of L. If we find that value of L for which $N_L \leq K$, but $N_{L-1} > K$, then L is the optimal load for the given K agents, and the LBAAP algorithm finds the nodes in the tree whose agents needs to be activated. The algorithm can be implemented efficiently using binary search to find the right value of L. Since binary search is used, there are at most $\log n$ calls made to LBAAP algorithm, which makes the running time of BAAP algorithm to be $O(n^2 \log n)$.

```
BAAP(T, r, K)
   LMin := max degree of any node in T
   nagents := LBAAP(T, r, LMin)
   if nagents <= K then
      return LMin
   LMax := total load of all nodes in T
   while LMin < LMax - 1 do {
      LMid := (LMin + LMax)/2
      nagents := LBAAP(T, r, LMid)
      if nagents > K then
         LMin := LMid
      else
         LMax := LMid
   return LMax
```

THEOREM 2. *Algorithm BAAP finds the smallest load for which K agents are sufficient.*

PROOF. There is no feasible solution for the problem with load smaller than the initial value of `LMin`, the degree of the tree. If the number of agents needed to keep the load under `LMin` is less than or equal to K, then the algorithm returns this smallest load as the solution. Otherwise, it maintains the invariant that the optimal load is between `LMin` and `LMax` and finds the optimal load using binary search. □

3. CONCLUSION

In this paper, we focused on the activation of NAK-suppressing agents on a multicast tree so as to alleviate the feedback implosion problem. We considered the generalized variations of the agent activation problem considered in [3], namely the LBAAP and the BAAP, and presented efficient algorithms for optimal activation of agents. We are currently investigating the complexity and algorithms for for the multi-tree version of the problems considered in this paper.

4. REFERENCES

[1] B. Cain and D. Towsley. Generic multicast transport services: Router support for multicast applications. In *NETWORKING*, pages 108–120, 2000.

[2] K. Calvert, J. Griffioen, and S. Wen. Lightweight network support for scalable end-to-end services. In *Proc. of ACM SIGCOMM*, 2002.

[3] O. Daescu, R. Jothi, S. Peri, B. Raghavachari, and K. Sarac. Load-balanced agent activation for reliable multicast. Technical report, Department of Computer Science, University of Texas at Dallas, 2003.

[4] B. Fenner. *Multicast traceroute (mtrace) 5.2*, September 1998. ftp://ftp.parc.xerox.com/pub/net-research/ipmulti/.

[5] S. Guha, A. Markopoulou, and F. Tobagi. Hierarchical reliable multicast: Performance analysis and placement of proxies. *Computer Communications.* to appear.

[6] S. Guha, A. Meyerson, and K. Munagala. Hierarchical placement and network design problems. In *Proc. of IEEE Symposium on Foundations of Comptuer Science*, 2000.

[7] K. Jain and V.V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual scheme and lagrangian relaxation. *Journal of the ACM*, 48:274–296, 2001.

[8] P. Ji, J.F. Kurose, and D. Towsley. Activating and deactivating repair servers in active multicast trees. In *Proc. Tyrrhenian International Workshop on Digital Communications*, 2001.

[9] R. Jothi and B. Raghavachari. Placement of proxy servers to support server-based reliable multicast. In *Proc. of IEEE International Conference on Networking*, 2004. to appear.

[10] P. Krishnan, D. Raz, and Y. Shavitt. The cache location problem. *ACM Transactions on Networking*, 8(5), October 2000.

[11] B. Li, M. Golin, G. Italiano, X. Deng, and K. Sohraby. On the optimal placement of web proxies in the internet. In *Proc. of IEEE INFOCOM*, 1999.

[12] C. Papadopoulos, G. Parulkar, and G. Varghese. LMS: A router-assisted scheme for reliable multicast. *ACM Transactions on Networking.* to appear.

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

[14] L. Qiu, V.N. Padmanabhan, and G.M. Voelker. On the placement of web server replicas. In *Proc. of IEEE INFOCOM*, 2001.

[15] R. Shah, R. Jain, and F. Anjun. Efficient dissemination of personalized information using content-based multicast. In *Proc. of IEEE INFOCOM*, 2002.

[16] J.K. Shapiro, J. Kurose, D. Towsley, and S. Zabele. Topology discovery service for router-assisted multicast transport. In *Proc. of Openarch*, 2002.

[17] A. Tamir. An $O(pn^2)$ algorithm for the p-median and related problems on tree graphs. *Operations Research Letters*, 19:59–64, 1996.