LOAD BALANCING FOR RELIABLE MULTICAST

Chao Gong, Ovidiu Daescu, Raja Jothi, Balaji Raghavachari, Kamil Sarac
Department of Computer Science
University of Texas at Dallas
Richardson, TX, USA
{cxg010700, daescu, raja, rbk, ksarac}@utdallas.edu

ABSTRACT
New applications emerge along with the rapid growth of the Internet. Many functionalities other than packet forwarding are being proposed to be added into routers for supporting those new applications. Those functionalities significantly improve the performances of the applications, but they incur overheads on routers at the same time. It is a thorny problem to reach a balance between the application performance and the functionality overhead. In this paper, we study that problem in the context of reliable multicast. We term it load-balanced agent activation problem (LBAAP). We regard the message implosion at multicast session source as the performance and the NAK message suppression agent at router as the functionality. We develop a polynomial running time algorithm for the LBAAP problem in single multicast tree case. We conjecture the LBAAP problem in multiple multicast tree case is NP-Hard and propose a heuristic for it.

KEY WORDS
Load balancing, Reliable multicast, NAK suppression.

1 Introduction

Reliable multicast provides reliable data transport from a single source to multiple receivers in the Internet. The chief challenge in reliable multicast is scalability. One main difficulty in making reliable multicast scalable is feedback message implosion at the multicast session source. As the number of receivers increases, their feedback messages back to the source will eventually overwhelm the computing resources and even the link bandwidth at the source.

The usual approach to ensure reliable delivery in reliable multicast protocols is Negative Acknowledgement (NAK). That is, receivers send out NAK messages to inform the source about packet loss. When a packet loss occurs close to the source, most of the receivers will detect the loss and send out NAK messages. A large amount of these NAKs can result in implosion at the source.

One common approach to avoid feedback message implosion is feedback message suppression. In this scheme, a special functionality, called NAK suppression agent, is set up on internal nodes in a multicast tree. NAK messages are only unicasted from a node to its nearest ancestor in the multicast tree on which the NAK suppression agent is activated. The ancestor node just forwards one NAK up the tree and suppresses all duplicate NAKs. NAK suppression agents are software modules located at routers. The NAK suppression agent can be turned on (activated) or turned off (deactivated) on a router for a given reliable multicast session passing through that router.

The NAK suppression agent is helpful for implosion prevention, which is a key issue in reliable multicast performance. But at the same time it incurs memory and processing overheads on routers and increases the transmission delay of NAK from receiver to source. The NAK suppression agent must store sequence number information for each outstanding NAK message in order to be able to suppress future duplicate NAKs. NAKs are extracted from IP fast forwarding path for more detailed process at the router where the NAK suppression agent is activated. The NAK suppression agent examines every received NAK to decide whether to forward it upstream or suppress it. Moreover, in order to eliminate the security vulnerability of false NAK [1], the NAK suppression agent needs to deploy some authentication mechanism which is time consuming. Those NAK processing procedures consume the computing resources on routers and result in longer transmission delay for NAKs being transferred from receivers to the source. Excessively long delay causes problems such as the possibility that a source will not receive any NAK until the corresponding data packet is deleted from the send buffer.

Given a multicast tree, the approach to activate NAK suppression agent is a difficult issue. On one hand, a trivial approach which activates NAK suppression agent on each internal node in a multicast tree precludes implosion at the source. But this approach leads to high total memory and processing overheads on those nodes, as well as long NAK transmission delay. On the other hand, activating NAK suppression agent on just a few nodes in a multicast tree reduces the total overheads and transmission delay. But this approach may fail to prevent implosion at the source, or even worse, may overload some nodes with excessive NAK messages and degrade the performance of all traffic through those nodes. If activating NAK suppression agent on a few nodes far away from the root, the source may still suffer implosion; if activating NAK suppression agent on a few nodes near the root, these nodes may be overwhelmed by a large volume of NAKs. Consequently the performance of the multicast session through these nodes may deteriorate.

In the context of multiple multicast trees, the situation becomes more complicated. If many multicast sessions pass through a router, the trivial approach mentioned above will activate NAK suppression functionality for all sessions, and the resultant memory requirement will overload the router. A naive solution to avoid the memory overload on routers is to deactivate NAK suppression functionality for one randomly chosen session on the overloaded router. However, such a solution has a deficiency that the casual selection of the session being "dropped" may trigger implosion at the source or overload other routers in that session with excessive NAKs.

Reaching a compromise between the performance of reliable multicast and the overheads of the functionality supporting reliable multicast is a complex problem. In this paper we explore that problem. Specifically, we study the Load-Balanced Agent Activation Problem (LBAAP):

> *How to determine the number and placement of NAK suppression agents in order to prevent implosion at multicast session sources by incurring the minimal total memory and processing overheads on routers and without overloading any router?*

As far as we know, this problem has not been addressed before. In this paper, we examine the LBAAP problem in different contexts and propose corresponding algorithms.

When considering to establish a new service, it is in the concern of Internet Service Providers (ISPs) to reduce their own costs such as keeping routers below certain loads or cutting down traffic. While reliable multicast has not been widely deployed yet, we believe that studying the relationship between the performance and overheads of reliable multicast will foster its deployment.

Along with the rapid growth of the Internet, more and more functionalities other than packet forwarding are being proposed to be added into routers for supporting new emerging applications. These applications include web caches [2], content delivery networks [3], multicast communication services [4], etc. In addition, the increasing rate of cyber-attacks in the Internet necessitates the development of effective defense mechanisms, most of which require additional supports from routers [5, 6]. Those functionalities incur non-trivial overheads on the routers hosting them. Our work is an instance of studying the balance between the application performance supported and the overheads introduced by the additional functionalities at routers.

The rest of this paper is organized as follows. In Section 2, we define and analyze the LBAAP problem. We survey related work in Section 3. In Section 4, we study the LBAAP problem in single tree case and in multiple tree case. In Section 5, we discuss the limitations of our proposed algorithms in deployment and possible extensions. Finally, we summarize our work in Section 6.

2 Problem Formulation

In this section, we introduce the multicast tree model, definitions, and assumptions used in this paper and describe the LBAAP problem in detail.

In NAK suppression scheme, a tree structure is constructed for each reliable multicast session. In that tree, the leaves represent the receivers, and the rest nodes, called internal nodes, represent the routers which are NAK suppression capable and on the multicast forwarding path. The root of the tree denotes the edge router connecting with the multicast session source. If we assume all of the routers in the network are NAK suppression capable, then the tree mentioned above overlaps the raw multicast tree, otherwise it overlays the raw multicast tree. Since the discussion in this paper is based on the tree structure defined above, we refer to it as multicast tree in the following sections.

The number of the children of a particular node in a multicast tree is the *degree* of that node. Assuming in *the worst case*, one single packet loss near the source makes every receiver emit a NAK message, and every NAK arrives at its destination. If a NAK suppression agent is activated on an internal node in a multicast tree, then the degree of that node means the minimal number of NAKs received by that agent in the worst case. For example, consider an internal node v hosting an agent a, the more agents activated on v's children which are internal nodes, the less NAKs received by agent a. If an agent is activated on each of the children of v that are internal nodes, then in the worst case, each child that is an internal node just forwards one NAK upstream and suppresses others, and each child that is a leaf sends out one NAK, the number of NAKs received by agent a is the same as the degree of node v. The *weight* of a node in a multicast tree is defined as follows: if the node is a leaf, its weight is 1; if the node is an internal node hosting an agent, its weight is 1; otherwise its weight is the sum of the weights of all of its children. We can regard the weight of a node as the number of NAKs sent up the tree from/via that node in the worst case, though it is not always true in an asymmetric network. If an agent is activated on an internal node v which does not host an agent, then the number of NAKs received by the agent on node v in the worst case is the old value of the weight of node v, and the new value of the weight becomes 1. The definition of weight is illustrated in Figure 1, where the number beside each node is the weight of the node. In Figure 1-a, no agent is activated in the tree, so $weight(v_2) = 1 + 1 = 2$ and $weight(v_1) = 2 + 1 = 3$. In Figure 1-b, an agent is activated on node v_2, so $weight(v_2) = 1$ and $weight(v_1) = 1 + 1 = 2$.

We assume that the multicast session source expects to receive just one NAK for a single packet loss. Hence a NAK suppression agent is always activated on the root of multicast tree in order to guarantee the source to receive no more than one NAK for a single packet loss. We also assume a NAK suppression agent supports just one multicast session, then multiple agents need activating on a router in order to handle multiple multicast sessions. The majority

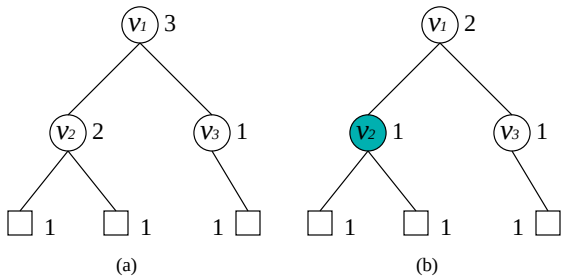

Figure 1. (a) No agent is activated on any internal node. (b) An agent is activated on internal node v_2.

of the memory overhead introduced by a NAK suppression agent is the amount of the memory recording the state information of NAKs, the amount of the memory storing agent's code itself is negligible. And the processing overhead on a router is not proportional to the number of agents on that router, but the number of arriving NAKs. Hence, assuming an agent to support single session or multiple sessions makes little difference to the LBAAP problem.

We define the *processing overhead introduced by an agent* to be the number of the NAKs received by the agent due to a single packet loss in the worst case defined above. The *processing overhead on a router* is the sum of the processing overheads of all of the agents activated on the router. We assume the *memory overhead introduced by an agent* is constant, and with value of 1. Hence the *memory overhead on a router* is the number of the agents activated on the router.

In a multicast tree, the sum of the memory overheads introduced by all NAK suppression agents is the number of agents, and the sum of the processing overheads introduced by all agents is the total number of the NAK messages received by those agents in the worst case. In the worst case, each receiver sends one NAK message to its nearest NAK suppression agent, and each agent except the one on the root forwards one NAK up to its nearest ancestor. Therefore the total processing overhead equals the sum of the number of receivers and the number of agents minus 1. So minimizing the number of agents is equivalent to minimizing the total memory and processing overheads on routers. The transmission delay of NAKs from a receiver to the source is proportional to the number of agents on the multicast forwarding path from the source to the receiver. Hence, in most cases, minimizing the number of agents means reducing the NAK transmission delay.

The Load-Balanced Agent Activation Problem (LBAAP) can be formulated as below: Given a memory load bound and a processing load bound on routers, the objective is to determine the minimal number of NAK suppression agents and locate the routers to activate those agents such that the memory and processing overheads on each router are not higher than the memory and processing load bounds, respectively.

The *memory load bound* on routers specifies the amount of the memory devoted for the NAK suppression functionality. In other words, how many NAK suppression agents can be activated on a router. The *processing load bound* on routers indicates the upper limit of the processing overhead on a router. The processing load bound reflects not only the amount of computing resources available for NAK processing on a router, but also the amount of bandwidth available for receiving NAKs.

Given certain memory and processing load bounds on the routers through which a multicast tree pass, minimizing the number of NAK suppression agents means minimizing the total memory and processing overheads and reducing the NAK transmission delay. In addition, locating routers to activate NAK suppression agent without exceeding the specified bounds of memory and processing capability prevents overloading routers and achieves load balancing among routers.

In this work, we assume that the multicast tree topology is already known and relatively stable. Typically ISPs have access to multicast tree information within their domains. Our work concerns NAK suppression agent activation within a domain. If a multicast tree spans several domains, our algorithms can be used to find an activation of agents in each domain, for the portion of the multicast tree that spans that domain. In addition, our *tracetree* proposal [7] can be used to collect multicast tree topologies efficiently. Finally, in most applications that use reliable multicast, such as one-to-many file transfer, the set of receivers is mostly static and is known to the source in advance. As a result, the collected tree topology is relatively stable during the application lifetime.

3 Related Work

In theoretic aspect, the LBAAP problem resembles two well-known graph theoretic problems: the k-median problem and the facility location problem. Given a graph with n nodes, the k-median problem is to select k out of n nodes as service centers so as to minimize the sum of the cost of each node accessing its nearest service center among those k selected nodes. Tamir [8] studies the k-median problem in a tree topology and gives an optimal algorithm. Li *et al.* [9] use a similar approach to optimally place web proxies in a tree topology with a web server at the root. Qiu *et al.* [10] study the same problem in a graph topology and propose various heuristics. Krishnan *et al.* [2] study the problem of optimal placement of web caches. Shah *et al.* [11] deal with the k-median problem in the context of content-based multicast.

In the facility location problem, besides the cost of accessing the nearest service center (facility), there is also a cost of building facility onto a node to make it become a service center, the objective is to find a solution (both the number and locations of the facilities) of the minimal total cost [12]. Guha *et al.* [13] introduce the Load Balanced Facility Location Problem wherein the constraint of having a minimum load on facility nodes is added to the original problem. They prove that problem is NP-Complete and present a constant factor approximation algorithm for it.

In practical aspect, Papadopoulos and Laliotis [14] in-

vestigate the performance of reliable multicast under various deployment strategies of the supporting functionality. In addition, lots of study has been done on the placement of other kinds of functionality agents in the context of reliable multicast [15, 16]. The objective of these works is to reduce the number of retransmissions, latency, and resource utilization.

4 Load-Balanced Agent Activation Problem

We examine the LBAAP problem in both single tree case and multiple tree case, and propose corresponding solutions.

4.1 LBAAP in Single Tree Case

In the context of a single multicast tree, at most one NAK suppression agent needs to be activated on a router. As long as the memory load bound is not set to be 0, the memory overhead introduced by an agent will not overload routers. So we don't consider the memory overload issue in single tree case. Because the degree of an internal node is the lower limit of the processing overhead introduced by an agent on the node, we assume the processing load bound is never set to be smaller than the degree of any internal node in the tree.

The LBAAP problem in single tree case can be defined as follows: Given a multicast tree T rooted at r, with leaves denoting receivers, and internal nodes denoting routers. The memory load bound on internal nodes is MB and $MB \geq 1$. The processing load bound on internal nodes is PB. The processing overhead on an internal node v is $pd(v)$. The goal is to select a set of internal nodes, R, to activate a NAK suppression agent on each node in R, satisfying the following conditions:

1. $r \in R$,
2. $\forall v \in R, pd(v) \leq PB$, and
3. the size of R is minimal.

We first show a canonical activation of agents, then present an algorithm to find such solution in linear time. Given an optimal activation of NAK suppression agent in a multicast tree with a processing load bound PB, we can transform this activation into another optimal activation which satisfies the following two conditions:

1. For any internal node v which hosts an agent but whose parent p does not, if we deactivate the agent on node v, and activate an agent on v's parent p, then the processing overhead introduced by the agent on p will exceed the processing load bound PB.
2. For any internal node v which hosts an agent and has siblings, the processing overhead on node v is not smaller than the weight of any of its siblings.

Given an optimal activation of agent in a multicast tree with a processing load bound PB, we can transform this activation as follows. First we move each agent as high in the multicast tree as possible without violating the load constraint. In other words, if an agent is activated on a node v but no agent is activated on v's parent p, then it must be the case that if we do not activate the agent on v but activate an agent on its parent p instead, the processing overhead on p, which is the same as the weight of p when no agent is activated on p, will exceed PB. Otherwise, we can do such transformation to obtain a new solution with the same number of agents. Second, we move each agent from the node where it is located to the sibling node whose weight is larger than the processing overhead introduced by the agent on current node, if possible. In other words, if we activate an agent on a child u of a node p, but not on another child v of p, then it must be the case that $pd(u) \geq weight(v)$. Otherwise, we do such transformation, then the processing overheads on all ancestor nodes of u remain same or decrease and we can obtain a new solution with the same number of agents.

```
                        /* T is multicast tree rooted at r */
LBAAP (T, PB) {         /* PB is processing load bound */
   nagents := 0
   pd(r) := ActivateAgent(r)
   activate NAK suppression agent on r
   nagents := nagents + 1
   return nagents
}

/* activate NAK suppression agent in the subtree rooted at v */
ActivateAgent(v) {
   if v is a leaf node then return weight(v)
   else {
      weight(v) := 0
      for each child u of v do
         weight(v) := weight(v) + ActivateAgent(u)
      while weight(v) > PB do {
         select an internal node child u of v with largest weight
         activate NAK suppression agent on u
         nagents := nagents + 1
         pd(u) := weight(u)
         weight(v) := weight(v) - weight(u) + 1
         weight(u) := 1
      }
      return weight(v)
   }
}
```

Figure 2. Algorithm. The global variable `nagents` counts the number of agents activated by the algorithm.

Figure 2 describes our proposed algorithm. Given a multicast tree T and a processing load bound PB, process multicast tree T from bottom to top, for every internal node v, if $weight(v) > PB$, select a child node, u, which is an internal node with the largest weight among all such nodes that are children of v, then activate NAK suppression agent on u. After that activation, $weight(v)$ reduces to $weight(v) - weight(u) + 1$, and $weight(u)$ becomes 1. Repeat above operation until $weight(v) \leq PB$, then go to the next node. At last, activate NAK suppression agent on the root of the multicast tree. The correctness of the algorithm follows from the argument above on the canonical activation of agents. The running time of the algorithm is $O(n \cdot d)$, n is the number of the internal nodes in the multicast tree, and d is the average degree of all internal nodes.

4.2 LBAAP in Multiple Tree Case

In this section we consider the LBAAP problem in the context of multiple multicast trees. Suppose multiple multicast

sessions pass through a router, if the memory load bound is smaller than the number of sessions, the router can't activate NAK suppression agent for every session, it has to choose some of the sessions and does not offer NAK suppression support to them. A random selection of the session being "dropped" from a router may make excessive NAK messages flow to the source or some upstream router in that session. That may trigger implosion at the source or overload the upstream router with excessive NAKs. So an intelligent selection of the sessions being served is a must. Since a NAK suppression agent is always activated on the root of multicast tree, we assume no router to be the root of more than MB trees, wherein MB is the memory load bound. In other words, there is not any edge routers connecting with more than MB multicast session sources.

The LBAAP problem in multiple tree case can be defined as follows: Given a graph $G = (V, E)$, with V denoting the set of nodes and E denoting the set of edges connecting the nodes. $V_h \subset V$ is the set of hosts and $V_r = V - V_l$ is the set of routers. $T = \{T_1, T_2, \ldots T_m\}$ is a set of m multicast trees in G. Each tree $T_i \in T$ is rooted at $r_i \in V_r$. The leaves (receivers) and internal nodes (routers) in each tree belong to V_h and V_r, respectively. The memory load bound on routers is MB, and the processing load bound on routers is PB. The memory overhead on a router v is $md(v)$, and the processing overhead on v is $pd(v)$. The goal is to select a set of nodes $R \subseteq V_r$, for $\forall v \in R$, activate $n_v (n_v \geq 1)$ agents on it, satisfying the following conditions:

1. $r_i \in R$, for $1 \leq i \leq m$,
2. $\forall v \in R, md(v) \leq MB$,
3. $\forall v \in R, pd(v) \leq PB$, and
4. $\sum_{v \in R} n_v$ is minimal.

As we will see in Section 4.3, even in a simplified model, the LBAAP problem in multiple tree case turns out to be much harder than the single tree case. We conjecture it is NP-Hard based on the fact that a minor variation of the multi-tree LBAAP problem is NP-hard. Therefore, we focus on computing a feasible solution rather than computing an optimal solution. We propose a heuristic to solve the LBAAP problem in multiple tree case.

First at all, we define an operation on tree topology, *removing node*. Removing a node v from a tree T means to remove node v from tree T, and turn all of the children of v to be the children of v's parent. The root of tree can not be removed.

The heuristic proceeds in two steps:

1. *Tree Selection:* Suppose the number of the multicast trees passing through a router v is N_v. For each node $v \in V_r$, if $N_v \leq MB$, then all N_v trees are selected as candidates which are supported by node v with NAK suppression agent. Otherwise we need to select MB out of N_v trees in the following way.
   (a) For each tree T_i passing through v, if v is the root of T_i, select tree T_i as a candidate.
   (b) After step (a), if the number of candidates is smaller than MB, then for each of the rest trees, remove node v from the tree. Arrange these trees into a list in decreasing order according to the degree of the parent of node v in each tree. Select trees from the beginning of the list as a candidate until the number of candidates increases to MB. For these trees selected as candidate, undo the operation of removing node v and restore them to their previous topologies.
2. *Processing Load Bound Assignment:* After making tree selection for each node $v \in V_r$, each tree remains the same topology or changes to a smaller topology by removing some internal nodes. We assign the processing load bound $L = \frac{PB}{MB}$ to each tree. Then apply the LBAAP algorithm for single tree case individually on each tree.

Step 1 makes sure each router supports no more than MB multicast trees, then the memory load bound is satisfied. Step 2 makes sure the processing overhead on each router is not larger than PB, then the processing load bound is satisfied.

When the processing load bound L is smaller than the degree of any internal node in a multicast tree, the LBAAP algorithm for single tree case can not produce a solution for agent activation in that tree. When the processing load bound on routers, PB, is set to be smaller, the processing load bound L assigned to each tree becomes smaller. When the memory load bound on routers, MB, is set to be smaller, more nodes are removed from each tree, the degrees of some nodes or all nodes in each tree become larger. So the LBAAP heuristic for multiple tree case can not produce a solution when the memory and processing load bounds on routers are set to be too small.

4.3 Hardness of Multi-tree LBAAP Problem

Currently, we do not know whether the LBAAP problem in multiple tree case is NP-Hard or not. However, we conjecture it is because a minor variation of the multi-tree LBAAP problem is NP-Hard.

Given a graph $G = (V, E)$, with V denoting the set of nodes and E denoting the set of edges. $V_h \subset V$ is the set of hosts and $V_r = V - V_l$ is the set of routers. $T = \{T_1, T_2, \ldots T_m\}$ is a set of m multicast trees in G. The leaves (receivers) and internal nodes (routers) in each tree belong to V_h and V_r, respectively. Each edge in E appears in some tree T_i and no edges are shared by any two different trees, but multiple trees may share the same node in V. Multiple NAK suppression agents are allowed on a node $v \in V_r$ and one agent can support multiple trees, but one tree passing through node v can be associated with at most one agent on v. There is not the memory load bound or processing load bound on nodes. Instead, we set the processing load bound L on agents. The objective is to activate as few number of agents as possible such that no agent has a processing overhead more than L. Clearly, this is a simpler variant of the multi-tree LBAAP problem.

Since the trees in T do not share edges, we can treat each tree $T_i \in T$ independently and compute an optimal agent activation for T_i using the LBAAP algorithm for single tree case. Assume this has been done for each tree in T. Then, on each node v, the number of agents is $k_v (0 \leq k_v \leq m)$ such that each agent supports just one tree. Since an agent may support multiple trees as long as its processing overhead does not exceed L, we would like to minimize the number of agents on each node by making single agent support multiple trees without violating the load constraint, hence minimize the total number of agents. This is an instance of the integer bin packing problem which is known to be NP-Hard [17].

5 Discussion

The NAK suppression agent activation algorithms proposed in this paper need multicast tree topologies as input. The deployment of these algorithms in practice is in centralized manner. That is, the algorithms are implemented at a central server. The server collects the multicast tree topologies, invokes the algorithms, then commands the relevant routers to activate NAK suppression agent. If a multicast group is dynamic, the multicast tree topology varies frequently, then the overheads of the algorithms on computing resources and network bandwidth will increase noticeably.

On one hand, as we mentioned previously, due to the semantics of the applications using reliable multicast, the tree topology of a reliable multicast session is relatively stable during the lifetime of the session. On the other hand, if the structure of a multicast tree changes partially, i.e., in a subtree, the agent activation algorithms can be applied on the subtree to compute a new activation of agents in the subtree. Combined with the old activation of agents in the rest of the tree, the new agent activation works for the whole tree. When applying the agent activation algorithms on a subtree instead of the whole tree, the overheads on both computation and bandwidth are reduced. For instance, a router v hosts an agent for a multicast session, when the agent receives more NAKs than its upper limit, router v sends a request to the server to ask for recomputing the activation of agents in the subtree rooted at v. The server will discover current topology of the subtree, invoke agent activation algorithms, and activate agents in the subtree.

Several areas remain to be addressed in future work. One is to generalize the LBAAP problem by allowing different nodes to have different load bounds, and develop corresponding algorithms. Another is to develop a NAK suppression agent activation approach which works in distributed manner.

6 Conclusion

In this paper, we have explored the relationship between the performance of reliable multicast and the overheads of the functionality supporting reliable multicast. In particular, We presented and studied the load-balanced agent activation problem (LBAAP). We regard the implosion as the performance and the NAK suppression agent as the functionality. We developed a polynomial running time algorithm for the LBAAP problem in single tree case, and proposed a heuristic for the LBAAP problem in multiple tree case since we conjecture it is NP-Hard.

Reliable multicast is one of many Internet applications that require the aid of various functionality agents on routers. Those functionality agents introduce non-trivial overheads on the routers hosting them. Reaching a compromise between the performance of those applications and the overheads incurred by the functionalities supporting those applications is worth delving.

References

[1] J. Gemmell, T. Montgomery, T. Speakman, N. Bhaskar, and J. Crowcroft, The PGM reliable multicast protocol, *IEEE Network*, 17(1), 2003, 16-22.

[2] P. Krishnan, D. Raz, and Y. Shavitt, The cache location problem, *IEEE/ACM Transactions on Networking*, 8(5), 2000, 568-582.

[3] B. Krishnamurthy, C. Wills, and Y. Zhang, On the use and performance of content distribution networks, *Proc. of SIGCOMM Workshop on Internet Measurement*, San Francisco, USA, 2001, 169-182.

[4] K. Almeroth, The evolution of multicast: From the MBone to interdomain multicast to Internet2 deployment, *IEEE Network*, 14(1), 2000, 10-20.

[5] A. Snoeren, C. Partridge, L. Sanchez, C. Jones, F. Tchakountio, B. Schwartz, S. Kent, and W. Strayer, Single-packet IP traceback, *IEEE/ACM Transactions on Networking*, 10(6), 2002, 721-734.

[6] A. Yaar, A. Perrig, and D. Song, SIFF: A stateless Internet flow filter to mitigate DDoS flooding attacks, *Proc. of IEEE Symposium on Security and Privacy*, Oakland, USA, 2004, 130-146.

[7] K. Sarac and K. Almeroth, Tracetree: A scalable mechanism to discover multicast tree topologies in the Internet, *IEEE/ACM Transactions on Networking*, 12(5), 2004.

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

[9] B. Li, M. Golin, G. Italiano, X. Deng, and K. Sohraby, On the optimal placement of Web proxies in the Internet, *Proc. of IEEE INFOCOM*, New York, USA, 1999, 1282-1290.

[10] L. Qiu, V. Padmanabhan, and G. Voelker, On the placement of Web server replicas, *Proc. of IEEE INFOCOM*, Anchorage, USA, 2001, 1587-1596.

[11] R. Shah, R. Jain, and F. Anjun, Efficient dissemination of personalized information using content-based multicast, *Proc. of IEEE INFOCOM*, New York, USA, 2002, 930-939.

[12] K. Jain and 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(2), 2001, 274-296.

[13] S. Guha, A. Meyerson, and K. Munagala, Hierarchical placement and network design problems, *Proc. of IEEE Symposium on Foundations of Computer Science*, Redondo Beach, USA, 2000, 603-612.

[14] C. Papadopoulos and E. Laliotis, Incremental deployment of a router-assisted reliable multicast scheme, *Proc. of International Workshop on Networked Group Communications*, Palo Alto, USA, 2000, 37-46.

[15] S. Guha, A. Markopoulou, and F. Tobagi, Hierarchical reliable multicast: Performance analysis and placement of proxies, *Computer Communications*, 26(10), 2003, 2070-2081.

[16] P. Ji, J. Kurose, and D. Towsley, Activating and deactivating repair servers in active multicast trees, *Proc. of Tyrrhenian International Workshop on Digital Communications*, Toarmina, Italy, 2001, 507-523.

[17] S. Skiena, *The Algorithm Design Manual* (New York, Springer-Verlag, 1998).