Optimal Allocation of Clients to Replicated Multicast Servers *

Zongming Fei, Mostafa H. Ammar, Ellen W. Zegura

Networking and Telecommunications Group
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
{fei,ammar,ewz}@cc.gatech.edu

GIT-CC-99-13

Abstract

Server replication is an approach that is often used to improve the scalability of a service. One of the important factors in the efficient utilization of replicated servers is the ability to direct client requests to the "best" server, according to some optimality criteria. Recently, there have been several proposals for multicast services in which a server delivers information to multiple clients simultaneously. Such proposals include multicasting of web content, multicast-based video services (on-demand and pay-per-view style services), multicasting of database content and broadcast disks. The goal of many of these proposals is to use multicast to enhance the ability of the service to handle a large number of clients economically. Multicast servers may be replicated for several reasons: to distribute the load among on-demand multicast servers, to balance the "feedback" load on the servers or on entities along the multicast tree from the servers, or to select the server that is at the root of the "best" multicast routing tree. In this paper we first give a definition of the *static multicast server selection problem*, in which we assume a set of static clients and multicast servers and consider how one might produce an optimal allocation of the clients to the servers. We propose a transformation method for deriving multicast server selection algorithms from the traditional multicast routing algorithms. To investigate the dynamic behavior of client join and leave and the cost incurred during the process, we next define the *dynamic multicast server selection problem*, in which the potential clients join and leave the multicast session dynamically, and the goal is to produce an optimal allocation of clients to servers with an emphasis on how this allocation behaves over time. We formulate the problem as a Markovian Decision Process (MDP) and analyze the tradeoff between the cost of the multicast tree(s) and the transition cost of establishing and removing links from the tree(s). We also explore the effect of join/leave frequency on optimal policy. Our analysis leads to two heuristics which we use to propose a selection algorithm. The algorithm uses a very simple join and leave strategy yet still can generate low cost trees. Our simulation compares the performance of our proposed algorithm with various other multicast server selection algorithms.

*This work is supported in part by a research grant from NSF under contract number NCR-9628379.

1 Introduction

Server replication is an approach that is often used to improve the scalability of a service. One of the important factors in the efficient utilization of replicated servers is the ability to direct client requests to the "best" server, according to some optimality criteria. Server replication and selection have been studied extensively in the context of unicast services [1-6], i.e., those in which a server communicates with clients in a point-to-point fashion. Typically, selecting among unicast servers is done with some goal in mind, e.g., minimizing client response time or balancing server load.

Recently, there have been several proposals for multicast services in which a server delivers information to multiple clients simultaneously. Such proposals include multicasting of web content [7-11], multicast-based video services (on-demand and pay-per-view style services) [12-16], multicasting of database content [17] and broadcast disks [18, 19]. The goal of many of these proposals is to use multicast to enhance the ability of the service to handle a large number of clients economically.

There are two types of multicast services:

1. *On-Demand Multicast Services:* These servers receive explicit requests from clients. The requests are aggregated according to some criteria to decide which data objects should be multicast.
2. *Continuous Multicast Services:* These servers continuously multicast content (without needing to receive explicit client requests).

Multicast servers may be replicated for several reasons:

1. To distribute the load among on-demand multicast servers (much in the same way that replication is used to distribute the load among unicast servers).
2. To balance the "feedback" load on the servers or on entities along the multicast tree from the servers. Examples of this feedback include acknowledgements required in many reliable multicast protocols [20, 21] or loss information required for adaptive video transmission [22, 23, 24].
3. To select the server that is at the root of the "best" multicast routing tree. From a receiver's perspective, joining a particular routing tree may be preferable because that tree may give better quality of service. For example servers that deliver data over a shorter (or lower delay) multicast routing path may be preferred. From a global perspective, having a receiver join a particular tree may reduce the overall cost of providing the service. For example the overall cost of providing a service may be reduced if a client is made to receive data from a server which is at the root of the closest *tree*; i.e., the tree requiring the shortest graft to reach the client.

In this paper, we study the multicast server selection problem. Our goal in this initial study is to consider some fundamental aspects of this problem. Towards that end we define two main problems:

- Static Multicast Server Selection: In this problem we assume a set of static clients and multicast servers and consider how one might produce an optimal allocation of the clients to the servers. For the static multicast server selection problem, we analyze the relationship between server selection and traditional multicast routing which seeks to establish one "optimal" tree from the server to all the receivers in the multicast group. When the multicast server is replicated, one needs to establish an "optimal" set of trees that reach all receivers with one server as a root for each tree. Our approach is to formulate several graph-theoretic problems that capture various multiple-tree optimization criteria and consider solution approaches that are based on single-tree multicast routing solutions.
- Dynamic Multicast Server Selection: In this problem we assume a fixed set of multicast servers but a dynamic set of clients with a given join/leave behavior. Again our focus is to produce an optimal allocation of clients to servers with an emphasis on how this allocation behaves over time. In this problem one needs to give consideration to the cost of joining and leaving a multicast group. We formulate the dynamic multicast server selection as a *Markovian Decision Process* (MDP) [25]. This allows us to factor in the stochastic join/leave behavior of receivers and to explore the tradeoff between the cost of the multicast tree(s) and the transition cost of establishing and removing links from the tree(s). Because of the limited scalability of MDP solution approaches, we are only able to use them to consider small dynamic multicast server selection problems. We use the properties of the optimal solutions generated from these small examples to develop a set of dynamic server selection heuristics.

The rest of this paper is structured as follows. In Section 2 we give definitions of the static multicast server selection problems and discuss solution approaches to them. In Section 3 we present a transformation method to solve a particular static selection problem, i.e., the minimum cost allocation problem. In Section 4 we give a model of the dynamic multicast server selection problem and in Section 5 we use MDP method to solve the problem and analyze several issues in dynamic server selection. In Section 6 we present a dynamic multicast sever selection algorithm based on the heuristics summarized from the analysis of solutions to the examples using MDP method. In Section 7 we present simulation results of various selection algorithms and the paper is concluded in Section 8.

2 Static Multicast Server Selection Problems and Solution Approaches

We use a conventional model where a network is represented as a graph $G = (V, E)$, where V denotes the set of nodes (representing hosts or routers) and E denotes the set of edges (representing logical connectivity). Edge $e \in E$ is of the form (v_i, v_j), where $v_i, v_j \in V$. We

assume G is a simple graph. We define the cost function on E as $C : E \rightarrow \Re$ and delay function on E as $D : E \rightarrow \Re$, where $\Re$ is the set of real numbers. A path between node s and d is a sequence of nodes $(v_0, \cdots, v_k)$ such that $(v_i, v_{i+1}) \in E$ $(0 \leq i \leq k-1)$ and $s = v_0, d = v_k$. The set of paths between nodes s and d in graph G is denoted as $P_G(s, d)$.

Allocation Problems Assume there are n servers $S = \{s_1, \cdots, s_n\}$ and m receivers $R = \{r_1, \cdots, r_m\}$ in graph $G = (V, E)$. We aim to determine n multicast trees $T_i = (V_i, E_i), (V_i \subseteq V, E_i \subseteq E, 1 \leq i \leq n)$, such that $s_i \in V_i$ and $\forall r_j \in R \; \exists T_i = (V_i, E_i)(r_j \in V_i)$.

We identify two major allocation problems based on their goals.

1. The *minimum cost allocation problem* is to find n multicast trees such that the total cost of tree links is minimum, i.e., minimize $\sum_{i=1}^{n} \sum_{e \in E_i} C(e)$. This is a generalization of the Steiner tree problem [26]. Our approach is to establish a relationship between the minimum cost allocation problem and the minimum Steiner tree problem. We propose a transformation method for deriving minimum cost allocation algorithms from corresponding algorithms for the minimum Steiner tree problem, which will be introduced in detail in the next section.

2. The *minimum delay allocation problem* is to find n multicast trees such that the average delay from servers to receivers is minimum. We first define the *distance* between node s and d in graph G as $dist_G(s, d) = \min_{p \in P_G(s,d)} \sum_{e \in p} D(e)$. Then the minimum delay allocation problem is to minimize

$$\sum_{i=1}^{n} \sum_{\substack{v_j \in V_i \\ v_j \in R}} dist_{T_i}(s_i, v_j)$$

   Since the total delay is minimized, the average delay is also minimized for a given m. This problem can be solved by using a shortest path tree algorithm [27]. We can simply allocate each receiver to its nearest server.

Problem Variations If we impose further constraints on the trees established, we may formulate several other allocation problems. In the following we describe some of them along with solution approaches that would utilize the solution to the two main problems discussed above.

1. The *balanced allocation problem* has constraints on the number of receivers allocated to each server. In general, assume there are n integers, $k_1, \ldots, k_n$ and $\sum_{i=1}^{n} k_i \geq m$. We require that the number of clients allocated to server s_i is less than or equal to k_i. Depending on the goal, the balanced allocation problem can be either *balanced minimum delay allocation problem* if the goal is to minimize the delay or *balanced minimum cost allocation problem* if the goal is to minimize the total cost.

   For this problem, we can establish trees from servers by setting up a path for one receiver at a time and checking the balancing constraint. For the balanced minimum delay allocation

problem, we can select the shortest among all $(n \times m)$ shortest paths from servers to receivers. If adding this path does not violate the balancing constraint, add it and remove the receiver and all the shortest paths leading to it from the list. Otherwise, select a next shortest path. This process continues until all the receivers have been added. For the balanced minimum cost allocation problem, we can select a path that has a smallest cost from receivers to any of the nodes already in the tree. If adding this path does not violate the balance constraints, add it and remove the receiver from the list. Otherwise select the next smallest cost path. The process continues until all the receivers have been added.

2. The *delay bounded minimum cost allocation problem* has a constraint on the longest delay from servers to receivers in the minimum cost allocation problem. Obviously, for this problem to have a solution the delay bound imposed by a client needs to be at least as long as the shortest delay from any of the servers to the client.

   To solve this problem, we can expand multicast trees in a similar way as in the balanced minimum cost allocation problem. Instead of checking the balancing constraint, we check the delay constraint.

3. If we impose the balancing constraint on the above problem, we get the *balanced delay bounded minimum cost allocation problem.* Depending on the problem parameters, this problem may not have a feasible solution. If a solution exists it may be found through expansion of the multicast trees in a similar way to the solution above where we check on both the balance and delay bound constraints.

3 Solving the Minimum Cost Allocation Problem

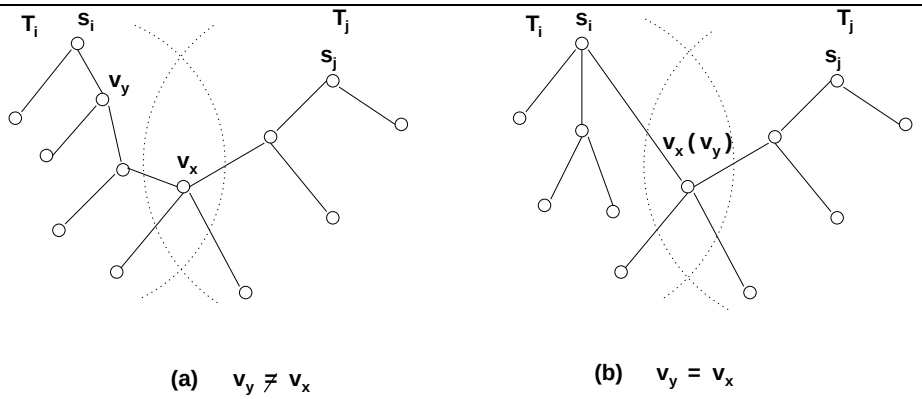
Here we focus on a solution to the minimum cost allocation problem which is a generalization of the minimum Steiner tree problem. First we show a disjointness property of the minimum cost allocation problem. Then we show an approach to transform the problem into a Steiner tree problem. This allows us to use any of the Steiner tree algorithms [27-31].

3.1 Property

Lemma Given a graph $G = (V, E)$ with $C(e) > 0$ for any $e \in E$, if $T_i = (V_i, E_i)$ $(1 \leq i \leq n)$ is a minimum cost allocation, then we have $V_i \cap V_j = \emptyset$ $(i \neq j)$. This is called the *mutual disjointness property.*

Proof. For a minimum cost allocation, if $V_i \cap V_j \neq \emptyset$, then there exists a node v_x that is in both trees, as shown in Figure 1. Consider the path from s_i to v_x. We delete the edge incident to s_i in this path (assume the other incident node of this edge is v_y, which can be a different node from v_x or the same node as v_x). Since all the nodes in the subtree rooted at v_y are still connected to node v_x, they are thus connected to s_j. All other subtrees of s_i are not affected by the above deletion. Include the subtree rooted at v_y as a part of tree T_j (with possible redundant edges removed to make it a tree). The resulting trees are still valid multicast trees, but the cost is less than the cost of the original because the cost of the deleted edge (s_i, v_y) is

FIGURE 1 An illustration of the proof of the disjointness property



greater than 0. Therefore this contradicts the fact that the original allocation is minimum cost allocation. So any two trees must be disjoint.

3.2 Transformation Method

We can reduce the multiple tree minimum cost allocation problem to a single tree routing (Steiner tree) problem with the following transformation method, assuming there are n servers $S = \{s_1, \cdots, s_n\}$ and m receivers $R = \{r_1, \cdots, r_m\}$ in graph $G = (V, E)$.

1. We add a virtual node s_0 to the graph and connect it to all servers with edges of cost 0. We get a new graph $G' = (V', E')$, where $V' = V \cup \{s_0\}$ and $E' = E \cup (\bigcup_{i=1}^{n} \{(s_0, s_i)\})$.

2. Solve the minimum cost allocation problem with a single server $\{s_0\}$ and m receivers $R = \{r_1, \cdots, r_m\}$. This is a minimum Steiner tree problem for G' with a required set of nodes $\{s_0\} \cup R$.

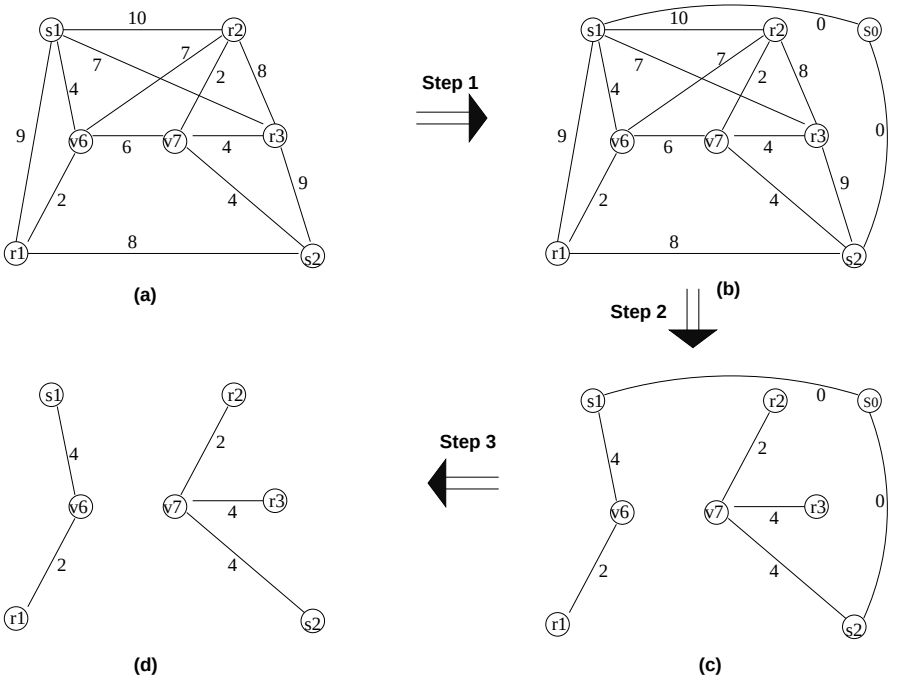
3. We get a tree T' from step 2. Remove any edge (s_0, s_i) $(1 \leq i \leq n)$ if it is in the tree. We get a forest T, which is the final allocation of receivers to servers.

Lemma If the cost function satisfies $C(e) > 0$ for any $e \in E$ and T' is a minimum Steiner tree for G' with a required set $\{s_0\} \cup R$, then the forest T (T' restricted to V) is a minimum allocation for G.

Proof: If T is not a minimum cost allocation, we assume that another allocation T_{new} has a smaller cost. We know all nodes in R are included in T_{new}. We connect s_0 to all s_i $(1 \leq i \leq n)$ with cost 0 and get T'_{new}. We get all required nodes in $\{s_0\} \cup R$ in T'_{new} and we know the cost of T'_{new} is equal to the cost of T_{new}, which in turn is smaller than the cost of T. Because T' has the same cost as T, the cost of T'_{new} is smaller than the cost of T'. This contradicts the fact that T' is a minimum Steiner tree in the graph G' with the required set $\{s_0\} \cup R$.

An example is given in Figure 2. The original graph is given in (a) with server set $S = \{s_1, s_2\}$ and receiver set $R = \{r_1, r_2, r_3\}$. The cost of each edge is shown in the figure. Step

FIGURE 2 An example of using KMB-based algorithm for server selection



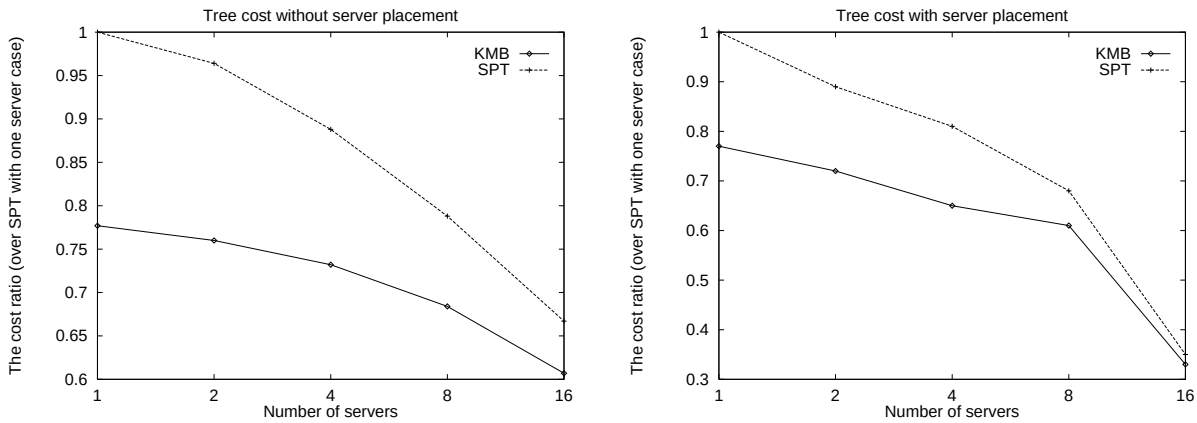
1 transforms the problem to a single tree routing problem (minimum Steiner tree problem) as shown in (b) with a required set $\{s_0, r_1, r_2, r_3\}$. Using the KMB algorithm [29] (step 2) we get a tree as shown in (c). By removing s_0 and edges (s_0, s_1) and (s_0, s_2), we get a solution to the multiple tree minimum cost allocation problem in (d). Note that the receivers do not necessarily select the nearest server, e.g., r_3 selects s_2 rather than s_1.

3.3 Numerical Results

In this section, we present some numerical results for static multicast server selection. Specifically we show how the cost of trees can be reduced by adding more servers. If we originally have a server set $S = \{s_1, \cdots, s_n\}$ and we add one more server s_{n+1}, the cost of trees generated with the new server set $S \cup \{s_{n+1}\}$ is less than or equal to that of trees generated with the original server set S. This is because we can build the same trees as before (thus, same cost) and make a possible improvement by using the new server.

In Figure 3 we show the results in two cases, one without server placement and one with server placement. We generate trees by using two transformed algorithms, one from the shortest path tree (SPT) algorithm and one from KMB. For brevity we still use SPT and KMB to denote the transformed algorithms. We generate a 400 node network topology using GT-ITM [33, 34] with 5% of the nodes as clients. In Figure 3(a) we assume servers are randomly placed at some nodes in the network. For each point in the plot, we experiment with 5 different server placements times 200 client placements. We get an average tree cost, which is normalized by computing the ratio over the cost generated by the SPT algorithm in the one server case. We

FIGURE 3 The effect on the tree cost by increasing the number of servers



(a) Random server placement

(b) Locality-based server placement

can see from the figure that the tree cost decreases as the number of servers increases in both algorithms. As expected the KMB algorithm performs better than the SPT algorithm, but they become closer as the number of servers becomes larger.

We can get more improvement if we observe the reference locality of clients and put servers in appropriate places, which is the server placement problem and beyond the scope of this paper. We only present a preliminary result in Figure 3(b). In GT-ITM we have the concept of stub domain, which, roughly speaking, is a set of nodes closely connected in the network. We make use of the concept to impose reference locality pattern in the network and require that nodes from a limited number of domains can be clients. We place servers in those stub domains that have clients and avoid putting two servers in the same domain. There are still quite a few domains that do not have a server. Placement of a server within a domain is randomized. We calculate an average tree cost and get the ratio over the average tree cost generated by the SPT algorithm with one server. We observe that the average tree cost decreases more rapidly than in the previous case for both KMB and SPT algorithms. The average tree cost with 16 servers is about 32% of the tree cost with one server for the SPT algorithm, almost half of the cost in the random placement case.

4 The Dynamic Multicast Server Selection Problem

In most multicast applications, the membership of receivers changes dynamically. A receiver can join or leave a multicast session at any time, which generally requires the addition (grafting) or removal (pruning) of paths from the multicast routing trees. This is normally accomplished through the use of a *multicast routing* protocol which uses the exchange of messages among multicast routers to establish or tear down the appropriate paths (See for example [35, 36]). Grafting and pruning, therefore, incurs a cost that is typically proportional to the length of the path being added or removed. This join/leave cost can be high if the receivers join and leave

frequently or if the paths being grafted or pruned are long. In this dynamic environment client allocation to multicast servers must, therefore, consider both the tree cost and the join/leave cost. The following is a model for the dynamic server selection problem that captures both the tree cost and join/leave cost.

Given a network $G = (V, E)$, we assume there are n servers $S = \{s_1, \cdots, s_n\}$ and m potential receivers $R = \{r_1, \cdots, r_m\}$. We assume that each receiver remains a member of the multicast session for a random period of time (the *time-in-session*) with the distribution $B(t)$ and mean $1/\mu$. Once it leaves the multicast session it re-joins after a random period (the *time-to-join*) with distribution $A(t)$ and mean $1/\lambda$. While a receiver is in a multicast session, it is receiving from exactly one server and belongs to the multicast group to which this server is transmitting. Note the distinction between *a multicast session* and *a multicast group*.

A cost rate function $F : E \rightarrow \Re^+$ is defined on the edges of the graph G. This represents the cost *per unit time* of using this edge as a part of a multicast tree. Another cost function $H : E \rightarrow \Re^+$ is defined on the the edges of G. This represents the cost of pruning or grafting this edge onto a multicast tree.

Each receiver r_j $(1 \leq j \leq m)$ can select any of the servers. At any given time, the *receiver state* can be described as $J =< j_1, \cdots, j_m >$, where $0 \leq j_k \leq n$ for $1 \leq k \leq m$. If $j_k = 0$, the receiver r_k is not in the multicast session at this time. If $j_k > 0$, the receiver r_k is in the multicast tree rooted at the server s_{j_k}. A transition of state occurs when a receiver joins or leaves the session. We say that a state J induces a set of trees $T_i = (V_i, E_i)$, $i = 1, \cdots, n$, that are subgraphs of G. The specific set of trees will be a function of the multicast routing in use. The set $\mathcal{T} = \{T_i;\ i = 1, \ldots, n\}$ completely specifies the state of the system. In the analysis to follow we assume that shortest path tree (SPT) routing is used to form the trees from the receiver state J [1]. Note that in general routing changes can result in system state changes without any corresponding changes to the receiver state. In our analysis we do not allow such changes and all system state changes occur as a result of receiver state changes.

There are two kinds of costs incurred in the dynamic join/leave process, *state occupancy* cost and *transition* cost.

- **State Occupancy Cost**

  The state occupancy cost is a measure of the cost incurred in operating the system in a particular state and is proportional to the time the process is in the state and the costs (from F) of the edges in the trees being used. More formally the state occupancy cost in state $\mathcal{T}$, which includes the trees $T_i = (E_i, V_i)$, $i = 1, \cdots, n$, is given by $\mathrm{OCost}(\mathcal{T}, t) = t \sum_{i=1}^{n} \sum_{e \in E_i} F(e)$, where t is the time spent in state $\mathcal{T}$.

- **State Transition Cost**

  The transition cost is incurred during the transition from one system state to another, that occurs as a result of a receiver joining or leaving. It is a function of the costs (derived

[1] Note that because of the possible routing ties, the combination of J and SPT routing does not unambiguously define the system state. For some network topologies (for example the one in Figure 2), however, it is possible to use the receiver state J in conjunction with the use of SPT routing to completely define the state of the system.

from H) of edges removed or added as a result of the transition between the two states. Consider the transition from state $\mathcal{T}_1$ to $\mathcal{T}_2$. This results in a change from a set of induced trees $T_{i1} = (E_{i1}, V_{i1})$ to another set of induced trees $T_{i2} = (E_{i2}, V_{i2})$, $i = 1, \cdots, n$. If we define the set of edges $L_i(\mathcal{T}_1, \mathcal{T}_2) = (E_{i1} \cup E_{i2}) - (E_{i1} \cap E_{i2})$, the total cost of this transition is given by $\mathrm{TCost}(\mathcal{T}_1, \mathcal{T}_2) = \sum_i \sum_{e \in L_i(\mathcal{T}_1, \mathcal{T}_2)} H(e)$.

Assume that the initial configuration of the tree is $\mathcal{T}_0$ and there are r transitions, with transition i changing the state from $\mathcal{T}_i$ to $\mathcal{T}_{i+1}$, where $i = 0, 1, 2, \cdots, r-1$. Assume that the duration in state $\mathcal{T}_i$ is t_i $(0 \leq i \leq r)$, and we have $t = \sum_{i=0}^{r} t_i$. The expected total cost of running the system for the t time units is given by

$$\mathrm{TotCost}(t) = E[\alpha \sum_{i=0}^{r} \mathrm{OCost}(J_i, t_i) + (1-\alpha) \sum_{i=0}^{r-1} \mathrm{TCost}(J_i, J_{i+1})] \tag{1}$$

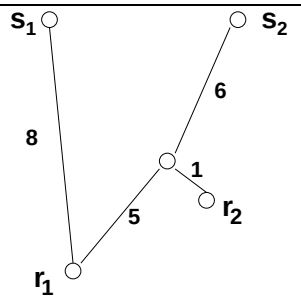
where α is a weight that can be used to change the relative cost of occupancy and transition.

If the transition from some receiver state $J =< j_1, \cdots, j_m >$ to another receiver state $J' =< j'_1, \cdots, j'_m >$ is triggered by receiver ℓ joining the session, then we know $j_\ell = 0$ because ℓ is not a member of any group before the join, and $j'_\ell > 0$ because ℓ must have selected some server and joined the multicast tree. If the transition is triggered by receiver ℓ leaving the session, then we know $j_\ell > 0$ and $j'_\ell = 0$. For any other receiver, the membership remains the same before and after the (join or leave) event. Therefore, if $j_k = 0$ then $j'_k = 0$ and if $j_k > 0$ then $j'_k > 0$ for all $k \neq \ell$. Whether j'_k is equal to j_k or not for $j_k > 0$ $(k \neq \ell)$ distinguishes two cases. One is the model *without switching*, in which a join/leave of one receiver will not change other receivers' selection of servers, i.e., $j'_k = j_k$ for all $j_k > 0$ $(k \neq \ell)$. The other is the model *with switching*, in which a join/leave of one receiver can change other receivers' selection of servers, i.e., j'_k can be different from j_k. Note that allowing server switching can lead to a lower cost of operation. Some applications may not allow for server switching, however.

In each state, a *decision* specifies a next state for each possible join/leave event. Assume the current receiver state is $J =< j_1, \cdots, j_m >$. If $j_\ell = 0$, the receiver ℓ may join a group. A decision in this case would indicate the server to which this new receiver should be allocated. In addition if our system allows switching, a decision would include any changes to the allocation of other receivers as a result of this particular join event occurring. If $j_\ell > 0$, the receiver ℓ may leave the group. A decision for this leave event would be specified if switching is allowed and would dictate the allocation changes to be made to remaining receivers. The new state J' is, therefore, determined by the decisions made, with different decisions leading to a different sequence of states over time.

A *policy* is a set of decisions, with one decision for each possible system state. The optimal policy is the one that minimizes the expected value of the cost per unit time, i.e., minimizes $\liminf_{t \to \infty} \frac{\mathrm{TotCost}(t)}{t}$.

FIGURE 4 The network setup



5 Solution to the Dynamic Multicast Server Selection Problem

5.1 Formulation as Markovian Decision Process

If receiver time-to-join and time-in-session distributions ($A(t)$ and $B(t)$) are exponential, then we can define a Markovian Decision Process (MDP) for the above model. Howard [25] developed a policy-iteration algorithm which is guaranteed to produce a policy that minimizes the cost for an MDP. The difficulty in applying the algorithm is that the state space grows exponentially with the number of receivers in the problem.

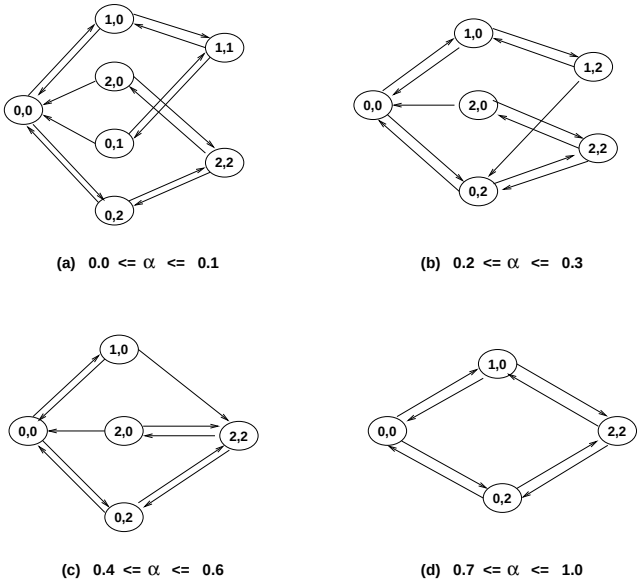
Our approach is to use the MDP method to investigate the dynamic nature of the multicast server selection problem for small systems and to develop solution heuristics that can be applied to large systems.

5.2 Tradeoff Between State Cost and Transition Cost

We now illustrate the interesting properties of the optimal policy using the simple network setup shown in Figure 4. There are two servers, s_1 and s_2, and two receivers, r_1 and r_2. Each edge is labeled with a single value that happens to be both the cost rate and the graft/prune cost. We will rely on the weight α in equation (1) to give us any further normalization of these costs. We assume the arrival rate and departure rate are 0.1. We can set α to 1 to get the best policy when we only consider the state occupancy cost. Similarly we get the best policy for cases when only the transition cost is considered by setting α to 0. We experiment with α from 0 to 1 with step 0.1 to see how the best policy changes as the weight of the state occupancy cost increases relative to the transition cost.

In Figure 5, we show the evolution of the best policy in the case where switching among servers is allowed. As previously defined, the state is in the form of a tuple, which gives the server selected by each receiver. For example, $< 2,1 >$ indicates that receiver r_1 selects server s_2 and receiver r_2 selects server s_1. A value of 0 means that the receiver is not in the multicast session. In the figure, we show only those states reachable from state $(0,0)$ by the policy. In all cases, receiver r_1 joins s_1 if neither r_1 nor r_2 is in the session, i.e., state $< 0,0 >$ to state $< 1,0 >$. Similarly receiver r_2 joins s_2, i.e., state $< 0,0 >$ to state $< 0,2 >$. The more interesting behavior occurs when the number of receivers in the session goes from 1 to 2 (due

FIGURE 5 The tradeoff between state occupancy cost and transition cost



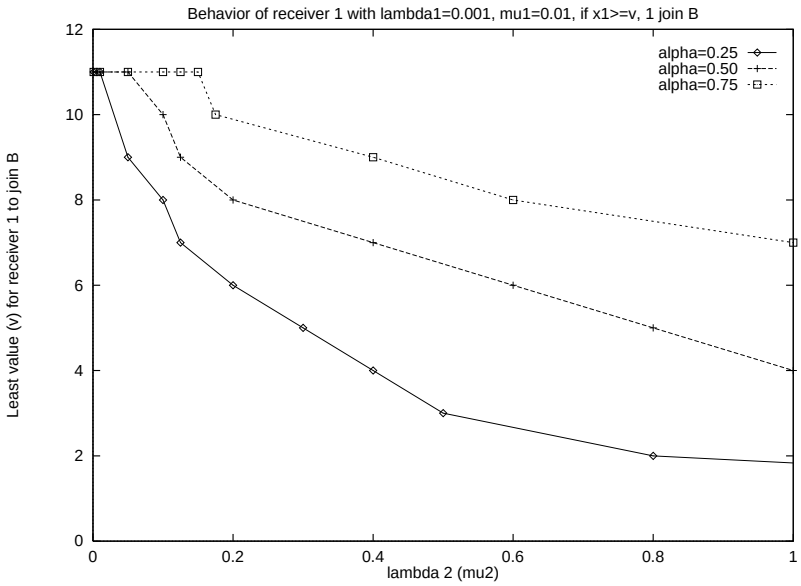
to a join) or from 2 to 1 (due to a leave). We show four cases based on different α values.

- (a) $0.0 \leq \alpha \leq 0.1$. This is the case in which the transition cost weight is extremely high and the state occupancy cost weight is small. If r_1 is connected to s_1 and r_2 arrives, r_2 will join s_1 because the join cost to s_1 is less than the join cost to s_2. When both r_1 and r_2 are connected to s_1, if either one leaves, the other remains in s_1. This is because changing a server when one leaves is extremely expensive. If r_2 is connected to s_2 and r_1 comes, r_1 will join s_2 because joining s_2 is less expensive. If both r_1 and r_2 are connected to s_2, if either one leaves, the other remains in s_2. We see in this case that a receiver never changes the selected server when another receiver joins or leaves.

- (b) $0.2 \leq \alpha \leq 0.3$. This is the case in which the transition cost weight is relatively high and the state occupancy cost weight relatively low. The difference of the best strategy from case (a) is that r_2 joins s_2 when r_1 is connected to s_1. The resulting state is $< 1, 2 >$. From this state, if r_1 leaves, r_2 remains with server s_2. This leads to a less expensive tree when the join sequence is (join r_1, join r_2, leave r_1) because now the state occupancy cost is more heavily weighted than the case (a). The price we pay is that when r_2 joins, it selects a more expensive transition by joining s_2 rather than joining s_1. Note that similar to (a), a receiver never changes the selected server in this case because the transition cost still has a relatively high weight.

- (c) $0.4 \leq \alpha \leq 0.6$. This is the case in which the transition cost weight is relatively low and the state occupancy cost weight relatively high. The difference between this case and the above two cases is that r_1 changes its selected server from s_1 to s_2 when r_2 joins. This is because the state occupancy cost weight is relatively high and the system would rather pay the cost of the transition of changing r_1 from s_1 to s_2 and get a tree (state $< 2, 2 >$) with less cost. If the state is in $< 2, 2 >$ and r_2 leaves, r_1 remains connected to s_2.

FIGURE 6 The least cost from receiver 1 to server A for receiver 1 to select B



- (d) $0.7 \leq \alpha \leq 1.0$. This is the case in which the transition cost weight is insignificant and the state occupancy cost weight is extremely high. The difference from case (c) is that when both r_1 and r_2 are connected to s_2 and r_2 leaves, r_1 will change from s_2 to s_1. This is because the transition cost weight is relatively low, and we try to get a minimum cost tree even when that requires some transition cost. In case (c) the transition cost weight is higher than this case, so we would rather get a slightly more expensive tree (11 instead of 8) and avoid paying a significant transition cost of changing r_1 from s_2 to s_1.

We can clearly see from the above analysis how the best policy evolves from transition-oriented to state-oriented by changing the parameter α.

5.3 Effect of Join/Leave Frequency on Optimal Policy

We now use the previous two receiver example to illustrate the effect of join/leave dynamics (as represented by λ and μ in our model) on the optimal policy. We set $\alpha = 0.5$ and arrival rate and departure rate for receiver r_1 to be $\lambda_1 = 10^{-3}, \mu_1 = 10^{-2}$. We first examine the case when arrival and departure rate for receiver r_2 are $\lambda_2 = 0.4, \mu_2 = 0.4$, respectively. We focus on the decision when the system is in state $< 0, 0 >$ and receiver r_1 joins the multicast session. If selecting the closest server is the criterion, then r_1 should select server s_1 since r_1 to s_1 is the shortest path and there are no other receivers in the system that can share cost. However, the best policy is for receiver r_1 to select s_2. This is because receiver r_2 has higher join and leave rate and has high probability of sharing the cost of the path. Also this policy may lead to a smaller join and leave transition cost for receiver r_2.

An interesting result is how short the path from r_1 to s_1 (denoted by x_1) needs to be before the best policy is for receiver r_1 to join s_1, given the values of λ_2 and μ_2. This is shown in

FIGURE 7 A server selection problem with 2 servers and 4 receivers

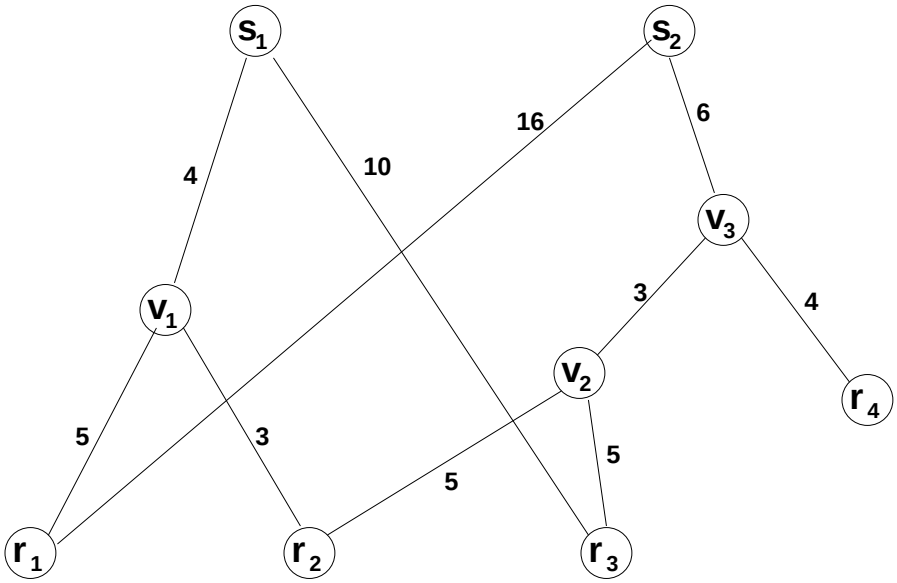


Figure 6. We change the the arrival and departure rate of r_2. We try different values for x_1 and find the critical value v so that r_1 will select server s_1 when $x_1 < v$. We vary λ_2 from 0.01 to 1.0. Let us focus on the case when $\alpha = 0.50$. We can see that the corresponding value of v changes from 11 to 4. For example, if $\lambda_2 = \mu_2 = 0.6$, then v is 6. That means, if $x_1 < 6$, receiver r_1 will select server s_1; if $x_1 \geq 6$, receiver r_1 will select server s_2, even if the path from r_1 to s_1 is much shorter than the path from receiver r_1 to server s_2. Note this selection strategy is used when there are no other receivers in the system. Similar behaviors have been observed when $\alpha = 0.25$ and $\alpha = 0.75$. Because a smaller α value means the transition cost has a heavier weight, r_1 begins to select server s_2 when x_1 has a smaller cost value.

5.4 A Larger Example

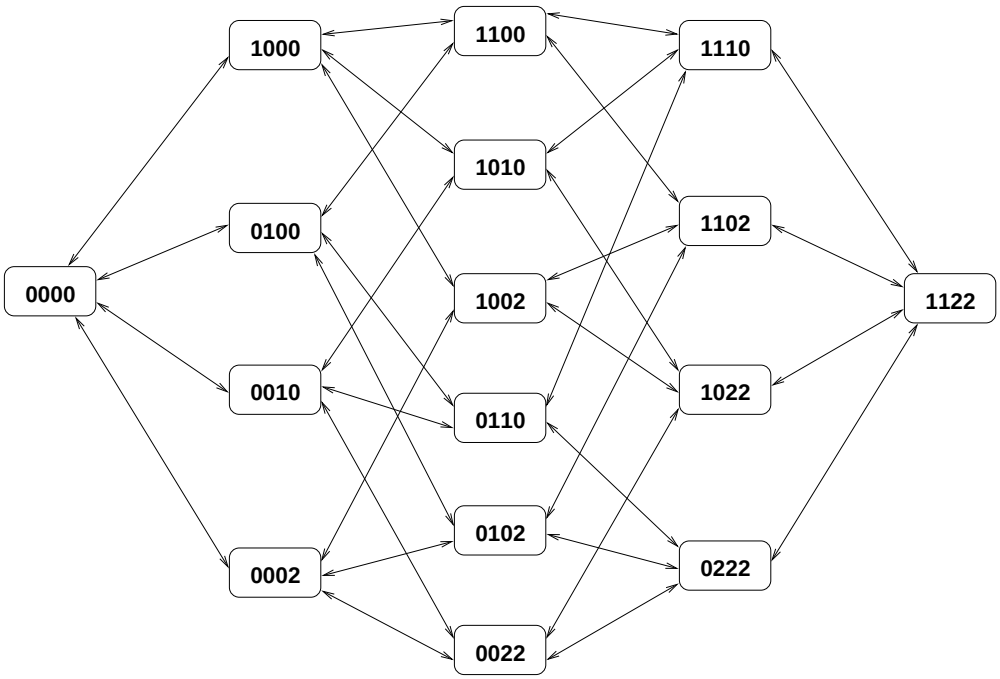
We now use the MDP approach to solve a selection problem with 2 servers and 4 receivers. Again our goal is to analyze the solution to the problem and identify heuristics that can be used to solve larger problems.

In Figure 7 we have two servers s_1, s_2, and four receivers r_1, r_2, r_3, r_4. Each edge is labeled with a single value that is both the occupancy cost rate and the graft/prune cost. We assume that the arrival rate is $\lambda = 10^{-4}$ and the departure rate is $\mu = 10^{-3}$ for all the receivers. We set the parameter $\alpha = 0.5$. In this example, we assume server switching is allowed. We use Howard's algorithm to obtain an optimal policy as shown in Figure 8, where states are in the form of tuples as previously defined.

We now consider the join and leave behavior of the receivers.

- **Join Behavior**

FIGURE 8 The solution to the server selection problem with 2 servers and 4 receivers



Receiver r_1 always selects s_1 and receiver r_4 always selects s_2. This is because selecting the other server will incur very high cost. The more interesting case is the behavior of receivers r_2 and r_3.

Consider the state transition from $< 0002 >$ to $< 0022 >$. Receiver r_3 will select s_2 if r_4 is already in the tree because the extra cost is 8 (from r_3 to v_3), which is smaller than selecting s_1 with an extra cost of 10 (from r_3 to s_1). Receiver r_3 will select s_1 if r_4 is not in the tree (e.g., the transition from $< 0000 >$ to $< 0010 >$) because the cost of selecting server s_1 is smaller than the cost of selecting server s_2. Similar is the case for receiver r_2. If r_3 is connected to server s_2, receiver r_2 will select server s_2 (e.g., the transition from $< 0022 >$ to $< 0222 >$), otherwise it will select server s_1 (e.g., the transition from $< 0000 >$ to $< 0100 >$).

An important assumption in our model is that we only consider the shortest path from a server to a receiver. So the candidate nodes that a receiver considers for server selection are those nodes in the shortest paths from all the servers to it. The above observation leads us to the following join heuristic: *A receiver will select the nearest in-tree node in its shortest paths to all servers and establish a path from that node to the receiver.*

- **Leave Behavior**

Leave is simple if we do not consider the effects of one receiver's leave on other receivers' selection. We can simply remove the path leading to this receiver that is not shared by other receivers. We now focus on how the tree changes for other receivers when one receiver leaves the tree.

Consider state $< 0022 >$, where receivers r_3 and r_4 are both connected to s_2. When receiver r_4 leaves the tree, r_3 changes from selecting server s_2 to s_1, leading to state $< 0010 >$. This is because r_3 can share links with r_4 in the original state, but the links are not shared once r_4 leaves.

Now we analyze state $< 0222 >$, where receivers r_2, r_3 and r_4 are connected to server s_2. When r_3 leaves, receiver r_2 changes from s_2 to s_1. We observe that r_2 shares a link with r_3. This indicates that when a receiver leaves, we should make an adjustment to those receivers sharing links with the departing receiver. We can reduce the cost by letting them re-select a server.

This leads us to the following leave heuristic: *When a receiver leaves, we find those receivers that share links with it and let them re-select servers.*

6 Dynamic Multicast Server Selection Algorithms

Our experience with the MDP-based solution leads us to some possible selection algorithms which we describe here. We first describe several join algorithms and then consider leave algorithms.

6.1 Join Algorithms

Nearest Server Algorithm. In the nearest server algorithm, a receiver will select the nearest server and join the multicast group associated with this server. In the worst case, this may require the grafting of the entire path from the server to the receiver. The problem with the approach is that it may still select the nearest server even if there is a nearby tree. Setting up a path from a nearby tree may add less cost than setting up a path from a server to the receiver. Therefore, the cost of the tree may be higher.

Nearest Tree Algorithm. In the nearest tree algorithm, a receiver joins the multicast group whose routing tree requires the shortest path to be grafted in order to join the group. It leads to a lower cost tree, but it is hard to implement. Usually a joining receiver needs to know all the nodes in the trees in order to select the nearest one.

Nearest Node Algorithm. Based on the join heuristic described in Section 5.4, we propose this new selection algorithm. Instead of letting a receiver select the nearest server or the nearest tree, we let the receiver select the nearest node among those that are in the shortest paths from servers to the receiver and already in the multicast trees.

Let Γ be the set of nodes in the shortest paths from servers to a receiver. Let Ω be the set of nodes that are already in the multicast trees. Select the nearest node $v \in \Gamma \cap \Omega$ and join the tree by establishing a shortest link from v to this receiver.

6.2 Leave Algorithms

No Adjustment. When a receiver wants to leave a multicast group, we can use the standard pruning method: if the receiver is not a leaf in the tree, simply remove the receiver from the group; if the receiver is a leaf in the tree, delete all the links that are not shared by other receivers.

With Adjustment. If we allow switching, we may be able to reduce the tree cost. The idea is to let all the receivers sharing edges with the deleted node re-select a server using a join algorithm. If the receiver is not a leaf, all the receivers in the subtree rooted at this node will re-join a group (re-select a server) using previous join algorithms. If the receiver is a leaf, we search the nodes towards the root of the tree until we locate an upstream node (v) satisfying one of the three conditions.

1) It is a server. No adjustment will be made.

2) It is a node that has at least two outgoing links (including the outgoing link connecting the departing receiver). We let each receiver in the subtree rooted at v re-select a server.

3) It is a receiver in the multicast group. We let v re-select a server.

In the nearest server algorithm, the receiver's choice is independent of the state of other receivers, thus leave adjustment will not reduce the cost. We can only use leave adjustment to reduce the cost for the nearest tree algorithm and the nearest node algorithm.

7 Performance Analysis

In this section, we describe results from a set of simulation experiments in which we study the performance of our proposed algorithm with various other multicast server selection algorithms.

7.1 Experimental Settings

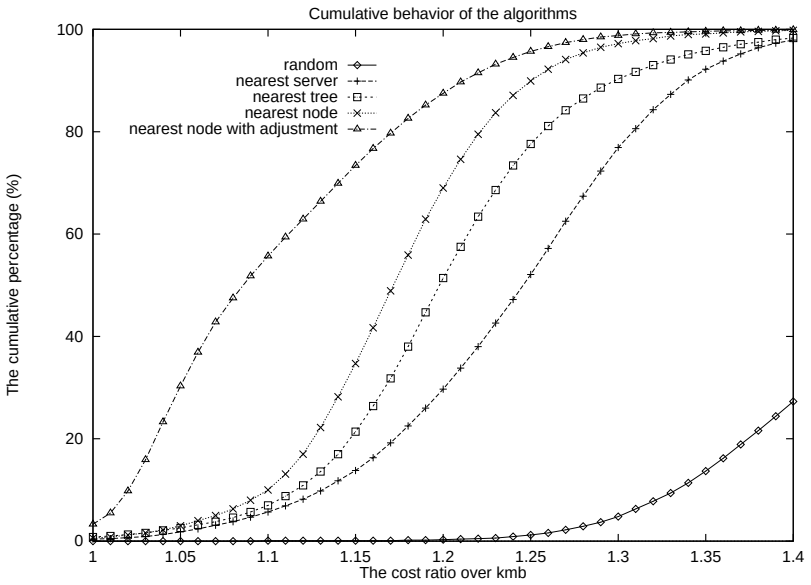
Network Topologies. We use the GT Internetwork Topology Models simulator [33, 34] to generate the network topology. We experiment with 1 to 16 servers in a network with 400 nodes. By setting the time-in-session and time-to-join for receivers, we can control the average number of receivers in the multicast session to be 5% to 30% of all the nodes, or 20 to 120.

Algorithms Compared. We compare the performance of five algorithms for multicast server selection.

1) Random selection. Every receiver randomly selects a server. A tree is constructed by the shortest path tree algorithm for each server.

2) Nearest server (SPT).

FIGURE 9 Cumulative distribution of the cost ratios



3) Nearest tree without adjustment.

4) Nearest node without adjustment.

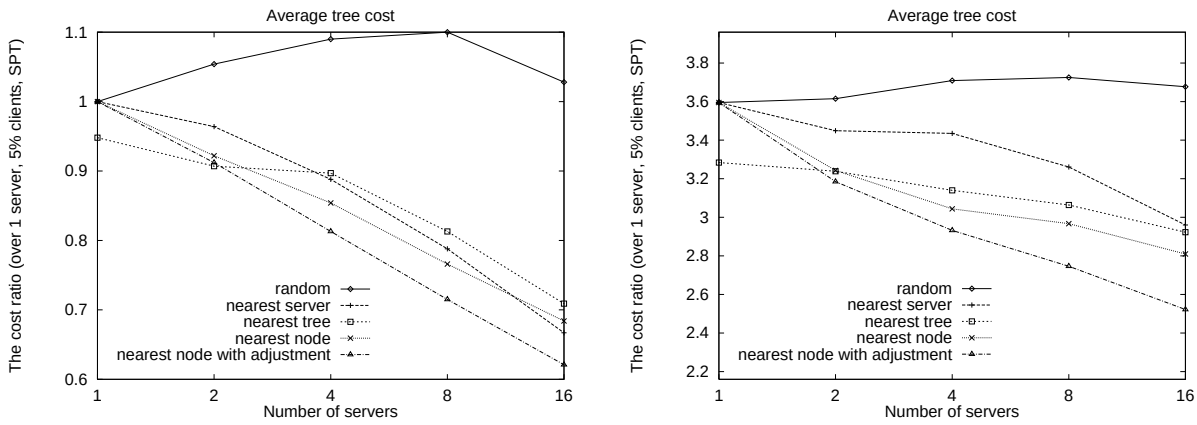
5) Nearest node with adjustment.

Metrics of Interest. Two basic metrics we use to compare the algorithms are the *average tree cost* and the *transition cost*. The network topology generated has a cost function defined on edges. All algorithms are based on the cost function. Another metric is the length of the path from the root of a tree to each receiver. We compare the average and maximum path length for the five algorithms.

7.2 Numerical Results

In the first experiment we have 2 to 16 servers in the network and the percentage of receivers in the multicast session changes from 5% to 30%. When a receiver joins or leaves the multicast session, we use the KMB-based algorithm described in Section 3.2 to construct a set of trees from scratch. We use the cost of these trees as the baseline cost for comparing the five algorithms. We calculate the ratio of tree cost in each case by each algorithm over the tree cost generated by the KMB-based algorithm in the same settings. The cumulative distribution of this ratio for each algorithm is shown in Figure 9. The x-axis is the ratio of the tree cost and the y-axis is the corresponding percentage. We find that random selection performs worst among the five algorithms. For example, the y-value is only 27% when x-value equals 1.4, which means, only 27% of the trees is within 1.4 times the tree cost generated by the KMB-based algorithm. All other algorithms are very close to 100% for this ratio. We can conclude that random selection for replicated multicast servers will result in very poor performance. For the other

FIGURE 10 The average tree cost



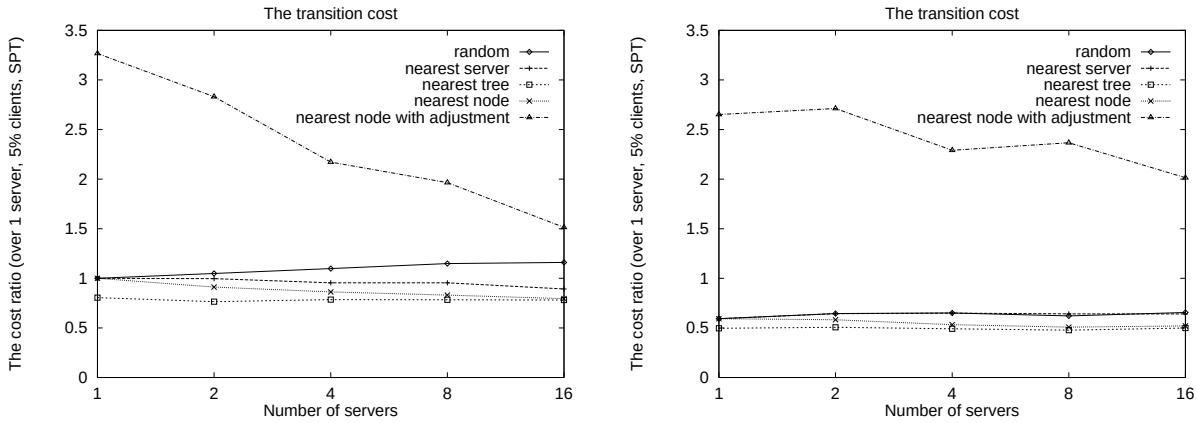
(a) 5% Clients (b) 30% Clients

four algorithms, the nearest node algorithms increase faster than either the nearest server or the nearest tree algorithm. The nearest node algorithm with adjustment performs best. For example, we can see that 78% of the trees generated by the algorithm is within 1.15 times the tree cost generated by the KMB-based algorithm, while only 20% of the trees generated by the nearest tree algorithm and less than 15% by the nearest server algorithm is within this range.

Figure 10(a) shows the average tree cost as a function of the number of servers. The average number of clients in the multicast session is maintained at the level of 5% of all the nodes in the network. We use the average tree cost generated by the SPT algorithm with 1 server as a comparison baseline and show the ratio of average tree cost over that value in each case. We can see the tree cost decreases as the number of servers increases, except in the random case. The nearest tree algorithm performs best when there is only one server. However when the number of servers increases, its average tree cost is larger than the nearest node algorithms. When the number of servers increases to 4 or more, it is worse than the nearest server algorithm. The random case generates the worst trees when the number of servers is more than 1. This is because the algorithm does not consider sharing between edges and may select a longer path that is not shared with any other receiver. Note that the 16 server case is better than 8 server case because the effect of increasing the number of servers outweigh the effect of random selection, but it is still worse than the 1 server case. The tree costs resulting from the use of the nearest node algorithms decrease faster than other algorithms. They outperform the nearest server algorithms when there are more than 1 server and are better than the nearest tree algorithm when there are 4 servers or more. The nearest server algorithm may add extra cost because of favoring the nearest server over a nearby in-tree node. The nearest tree algorithm may lead to an unusually long paths in the tree after several joins/leaves. The nearest node algorithms are a compromise between these two extremes and result in better performance in the multiple server cases.

Figure 10(b) shows the case when the average number of clients in the session is maintained

FIGURE 11 The join/leave cost



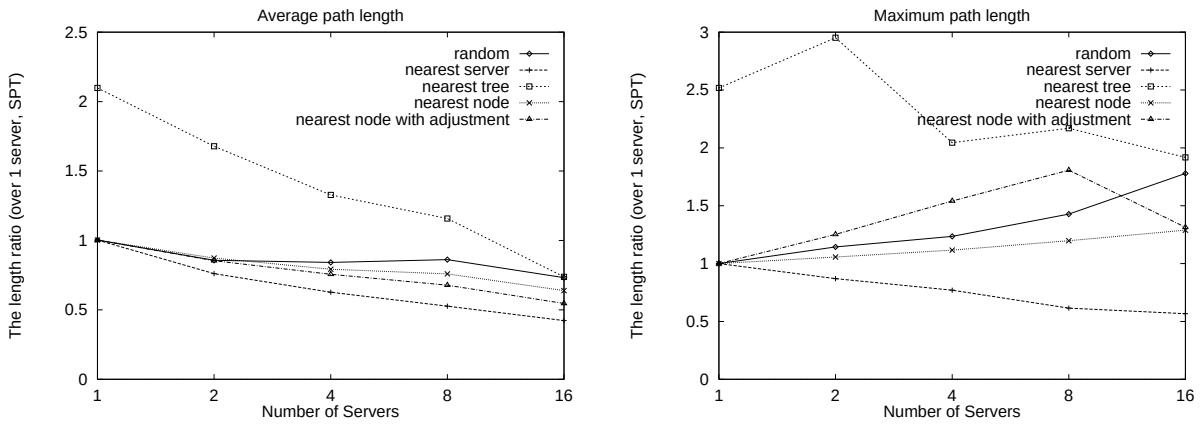
(a) 5% Clients (b) 30% Clients

at the level of 30% of all the nodes in the network. To make this comparable with the 5% client case, we use the ratio over the cost generated by the SPT algorithm in the 1 server case with 5% clients. By comparing the two plots, we can see that the average tree cost does not increase linearly with the average number of clients in the session because of sharing of cost among clients. Another observation is the effect of the average number of clients in the session on how the average tree cost changes with the number of servers. When there are 30% clients, the average tree costs do not decrease so fast as in the 5% client case. This is because the degree of sharing is already high in the single server case when there are a lot of clients and increasing the number of servers does not improve the performance as much. Another difference is that the nearest tree algorithm performs better than the nearest server algorithm. This is because we have a better chance of sharing for the nearest tree algorithm and the effect of unusually long paths is reduced due to the larger number of receivers in the case.

Figure 11(a) shows the join and leave cost when the average session size is 5% of nodes. We get the average cost over time and use the ratio over the cost induced by the SPT algorithm in the 1 server case. Generally the join and leave cost decreases as the number of servers increases, except in the random case. For the random algorithm, adding more servers does not help to decrease the join/leave cost. The join/leave costs of the five algorithms are rather similar except for the nearest node algorithm with adjustment. This is because the algorithm not only adds/removes a path for the join/leave client, it may change the path for other clients in the tree as well.

Figure 11(b) shows the case where the average session size is 30% of nodes. Again we use the ratio over the cost induced by the SPT algorithm in the 1 server case with 5% clients. The difference between costs with different number of servers is smaller in this case. This is because there is a higher chance that nearby nodes are already in the tree and the effect of the number of servers is lessened. This also leads to a smaller average join/leave cost than in the 5% client case. The only exception is the algorithm with adjustment, which has higher join/leave cost

FIGURE 12 The length from servers to clients



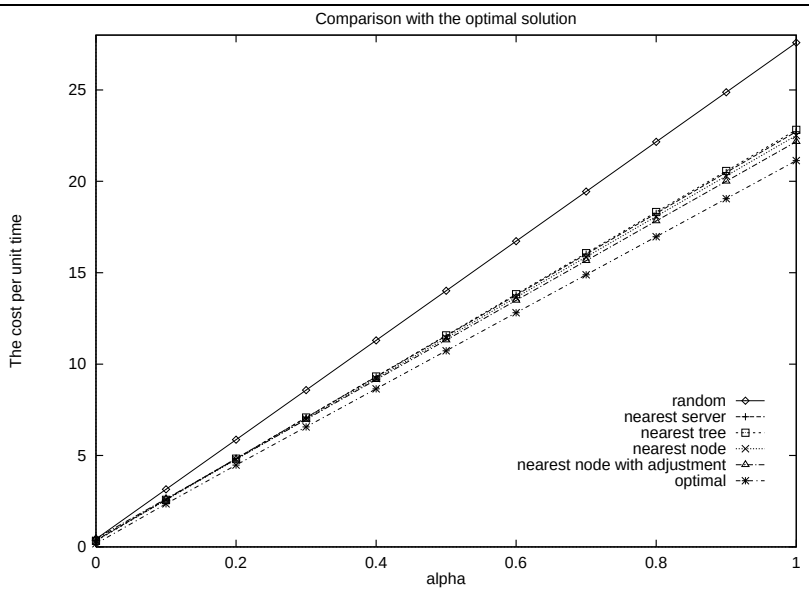
(a) Average length (b) Maximum length

for 4 or more servers than in the 5% client case. This can be explained by the fact that a larger number of clients in the session results in adjusting more clients in each join and leave.

Next we measure the length of the shortest path from each client to the root of the tree to which it belongs. We get an average path length for all clients over time, which is shown in Figure 12(a). Because the number of clients in the multicast session has little effect on the average path length, we do not show different plots for 5% and 30% client cases as in the previous experiments. We also use the ratio over the average length generated by the SPT algorithm with 1 server. Generally the average path length decreases as the number of servers increases. As expected, the nearest server algorithm performs best. The nearest tree algorithm is worst because of the effect of the unusually long paths. This effect is reduced when the number of servers increases. Figure 12(b) shows the maximum path length. The maximum path length decreases as the number of servers increases only in the nearest server algorithm. An interesting phenomenon is that the maximum path length is sometimes larger when we have more servers. This can be explained by the fact that the probability of getting a very long path by those algorithms is increased when the number of servers is increased.

In our last experiment, we compare the performance of the algorithms with the MDP method described in Section 5.1. We use the network topology shown in Figure 7. We vary the value of α from 0.0 to 1.0. The arrival rate of clients is 0.01 and the departure rate is 0.005. In each case, we calculate the sum of the state cost and the transition cost and get the cost per unit time for each algorithm. We use the MDP method to calculate the cost per unit time in the optimal solution. Because the state cost per unit time is larger than the transition cost per unit time, the total cost per unit time increases as we put more weight on the state cost by increasing α. Except for the random algorithm, the performance of all other four algorithms is very close to the optimal method. This is partly due to the small network topology that limits the extent of variations among these algorithms. If we explore the subtle difference between them, the nearest tree is the worst and the nearest node with adjustment is the best.

FIGURE 13 Comparison with the optimal solution by MDP



8 Concluding Remarks

In this paper, we define two categories of multicast server selection problems: static selection and dynamic selection. For the static multicast server selection problem, we present a transformation method for deriving algorithms from the traditional multicast routing algorithms. For the dynamic multicast server selection problem, we formulate it as a Markovian Decision Process. We explore the tradeoff between the cost of the multicast trees and the transition cost of establishing and removing links from the trees. We analyze the properties of the optimal solutions generated from these small examples to develop a set of server selection heuristics. Our simulation shows that selection algorithms based on these heuristics can perform better than other simple algorithms, such as nearest tree and nearest server algorithms.

This has been a mostly theoretical study focusing on some fundamental aspects of the multicast server selection problem. In future work we will consider more practical aspects of this problem with the objective of developing a framework and a set of protocols that can be used by clients for server selection. We will also consider the issue of server placement and how that affects the performance of the protocols used for server selection.

References

[1] S. Bhattacharjee, M. H. Ammar, E. W. Zegura, V. Shah, and Z. Fei, "Application layer anycasting," in *Proceedings of INFOCOM'97*, 1997.

[2] Z. Fei, S. Bhattacharjee, E. Zegura, and M. Ammar, "A novel server selection technique for improving the response time of a replicated service," in *Proceedings of INFOCOM'98*, 1998. San Francisco, California.

[3] R. L. Carter and M. E. Crovella, "Server selection using dynamic path characterization in wide-area networks," in *Proceedings of INFOCOM'97*, 1997.

[4] S. Seshan, M. Stemm, and R. H. Katz, "SPAND: Shared passive network performance discovery," in *Proc 1st Usenix Symposium on Internet Technologies and Systems (USITS '97)*, Dec. 1997. Monterey, CA.

[5] A. Myers, P. Dinda, and H. Zhang, "Performance characteristics of mirror servers on the Internet," in *Proceedings of INFOCOM'99*, Mar. 1999. New York.

[6] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan, "Service location protocol," *RFC 2165*, June 1997.

[7] R. Clark and M. Ammar, "Providing scalable web service using multicast delivery," *Computer Networks and ISDN Systems Journal*, vol. 29, pp. 841–858, 1997.

[8] K. C. Almeroth, M. Ammar, and Z. Fei, "Scalable delivery of web pages using cyclic best-effort (UDP) multicast," in *Proceedings of INFOCOM'98*, 1998. San Francisco, California.

[9] M. Ammar, K. Almeroth, R. Clark, and Z. Fei, "Multicast delivery of web pages or how to make web servers pushy," in *Proceedings of the Workshop on Internet Server Performance*, June 1998. Madison, Wisconsin.

[10] J. Nonnenmacher and E. Biersack, "Asynchronous multicast push: AMP," in *Proceedings of ICCC'97*, pp. 419–430, Nov. 1997. Cannes, France.

[11] Rodriguez and E. Biersack, "Continuous multicast push of web documents over the Internet," *IEEE Network Magazine*, vol. 12, pp. 18–31, Mar-Apr 1998.

[12] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling policies for an on-demand video server with batching," in *Proceedings of ACM Multimedia 94*, pp. 15–23, 1994.

[13] K. A. Hua and S. Sheu, "Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems," in *Proceedings of ACM Sigcomm'97*, 1997.

[14] L. Gao, J. Kurose, and D. Towsley, "Efficient schemes for broadcasting popular videos," in *Proceedings of NOSSDAV'98*, 1998.

[15] K. Almeroth and M. Ammar, "On the use of multicast delivery to provide a scalable and interactive video-on-demand service," *Journal of Selected Areas of Communication*, August 1996.

[16] K. Almeroth and M. Ammar, "The interactive multimedia jukebox (IMJ): A new paradigm for the on-demand delivery of audio/video," in *Proceedings of the 7th WWW conference*, April 1998. Brisbane, Australia.

[17] S. Mahajan, M. Donahoo, S. Navathe, M. Ammar, and S. Malik, "Grouping techniques for update propagation in intermittently-connected databases," in *Proceedings of the 1998 IEEE Conference on Data Engineering*, Feb. 1998. Orlando, Florida.

[18] R. Alonso, M. Franklin, and S. Zdonik, "Broadcast disks: Data management for asymmetric communication environments," in *Proceedings of the ACM SIGMOD*, May 1995. San Jose, CA.

[19] M. Franklin and S. Zdonik, "Balancing push and pull for data broadcast," in *Proceedings of the ACM SIGMOD*, May 1997. Tuscon, AZ.

[20] S. Floyd, V. Jacobson, C.-G. Liu, S. Mccanne, and L. Zhang, "A reliable multicast framework for light-weight sessions and application level framing," *IEEE/ACM Transactions on Networking*, Nov 1996.

[21] J. C. Lin and S. Paul, "RMTP: A reliable multicast transport protocol," in *Proceedings of INFOCOM'96*, March 1996. San Francisco, California.

[22] J. C. Bolot, T. Turletti, and I. Wakeman, "Scalable feedback control for multicast video distribution in the Internet," in *Proceedings of ACM SIGCOMM'94*, 1994.

[23] S. Y. Cheung, M. Ammar, and X. Li, "On the use of destination set grouping to improve fairness in multicast video distribution," in *Proceedings of INFOCOM'96*, March 1996. San Francisco, California.

[24] X. Li and M. Ammar, "Video multicast over the Internet," *IEEE Network Magazine*, April/May 1999.

[25] R. Howard, ed., *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, 1960.

[26] E. N. Gilbert and H. O. Pollak, "Steiner minimal trees," *SIAM Journal of Applied Mathematics*, vol. 16, no. 1, pp. 1–29, 1968.

[27] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[28] L. Kou, G. Markowsky, and L. Berman, "A fast algorithm for Steiner trees," *Acta Informatica*, vol. 15, pp. 141–145, 1981.

[29] A. Zilikovesky, "An 11/6-approximation algorithm for the network Steiner problem," *Algorithmica*, vol. 9, pp. 463–470, 1993.

[30] B. M. Waxman, "Routing of multipoint connections," *IEEE Journal of Selected Areas in Communications*, vol. 6, pp. 1617–22, December 1988.

[31] M. Doar and I. Leslie, "How bad is naive multicast routing?," in *Proceedings of INFOCOM'93*, 1993.

[32] F. Bauer and A. Varma, "Distributed algorithms for multicast path setup in data networks," *IEEE/ACM Transactions on Networks*, vol. 4, pp. 181–191, April 1996.

[33] E. W. Zegura, K. Calvert, and S. Bhattacharjee, "How to model an internetwork," in *Proceedings of INFOCOM'96*, 1996. San Francisco, CA.

[34] K. Calvert, M. Doar, and E. W. Zegura, "Modeling Internet topology," *IEEE Communications Magazine*, June 1997.

[35] S. Deering and D. Cheriton, "Multicast routing in datagram internetworks and extended LANs," *ACM Transactions on Computer Systems*, vol. 8, pp. 85–110, May 1990.

[36] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C.-G. Liu, and L. Wei, "The PIM architecture for wide-area multicast routing," *IEEE/ACM Transactions on Networking*, April 1996.