Multicast Server Selection: Problems, Complexity and Solutions

Zongming Fei, Mostafa Ammar, Ellen Zegura

Abstract—

We formulate and investigate fundamental problems that arise when *multicast servers*, that deliver content to multiple clients simultaneously, are *replicated* to enhance scalability and performance. Our study consists of two parts. First, we consider the problem under the assumption that the multicast clients are *static* for the duration of the multicast content distribution session. In this context, we examine two models for server behavior: fixed-rate servers, which transmit at a constant rate, and rate-adaptive servers, which adapt their transmission rate based on network conditions and/or feedback from clients. In both cases, we show that general versions of the client assignment problems are NP-complete. We then develop and evaluate efficient algorithms for interesting special cases, as well as heuristics for general cases. Second, we consider the case in which the set of clients changes *dynamically* during the multicast content distribution session. We again consider both fixed-rate and rate-adaptive servers. We formulate the problem as a Markov Decision Process, capturing the costs associated with trees, as well as the transition costs to dynamically change the trees. We use the properties of optimal solutions for small examples to develop a set of dynamic server selection heuristics.

Keywords— **Multicast Server Selection, Anycasting, Routing**

I. Introduction

With the explosive growth of the Internet, networked services must potentially handle a large number of clients. An increase in service demand often leads to system performance implications: degraded performance as perceived by the client, possible rejection of the request due to server overload, and poor performance observed by other network connections due to hot spots. To help maintain performance under higher demands, there have been several proposals for multicast services in which a server delivers information to multiple clients simultaneously (e.g., [1], [2], [3]). Through the aggregation of request processing at the server and the aggregation of bandwidth requirements in the network, a multicast service can reduce the load on the server and network and, therefore, improve the performance perceived by the clients.

Multicast servers may be replicated for several reasons: 1) To distribute the load among *on-demand* multicast servers, which receive explicit requests from clients and aggregate them according to some criteria to decide which data objects should be multicast [3]. 2) To distribute the "feedback load" on the servers or entities along the multicast tree from the servers. Examples of this feedback include acknowledgements required in many reliable multicast protocols [4] or loss information required for adaptive video transmission [5]. 3) To improve quality of service provided to clients and reduce the delivery cost.

Multicast server replication, however, needs to be coupled with an appropriate *server selection* (or *client assignment*) mechanism that determines which clients are served by which server. In this paper we formulate and investigate fundamental problems related to the assignment of clients to replicated multicast servers. This investigation lays the foundation for further development of practical and deployable protocols.

Our study consists of two parts. First, we consider the problem under the assumption that the multicast clients are *static* for the duration of the multicast content distribution session. In this context, we examine two models for server behavior: fixed-rate servers, which transmit at a constant rate, and rate-adaptive servers, which adapt their transmission rate based on network conditions and/or feedback from clients. In both cases, we show that general versions of the client assignment problems are NP-complete. We then develop and evaluate efficient algorithms for interesting special cases, as well as heuristics for general cases.

Second, we consider the case in which the set of clients changes *dynamically* during the multicast content distribution session. We again consider both fixed-rate and rate-adaptive servers. We formulate the problem as a Markov Decision Process, capturing the costs associated with trees, as well as the transition costs to dynamically change the trees. We use the properties of optimal solutions for small examples to develop a set of dynamic server selection heuristics.

Server replication and selection has been extensively studied in the unicast environment [6], [7], [8], [9], [10], [11]. We are not aware of any work prior to ours that addresses the question of how to allocate clients to replicated multicast servers. Our work is related to the replicated stream approach proposed for dealing with heterogenity of receivers in multicast sessions [12], [13], [14], [5]. This previous work deals with replicated streams from a *single* multicast server, whereas our work considers server replication that naturally leads to stream replication but from *multiple* multicast servers. Our examination of rate-adaptive servers is motivated by proposals for adapting the rate of multicast video to match client reception capability [15], [16], [5], [17].

The remainder of the paper is structured as follows. Section II describes the network model. In Section III we consider static server selection problems. In Sections IV and V we consider dynamic server selection problems for fixed-

Zongming Fei is with the Department of Computer Science, University of Kentucky, Lexington, KY. E-mail: fei@cs.uky.edu. Mostafa Ammar and Ellen Zegura are with the College of Computing, Georgia Tech, Atlanta, GA. E-mail: {ammar,ewz}@cc.gatech.edu.

rate and rate-adaptive servers, respectively. We conclude in Section VI.

II. Model

We use a conventional model where a network is represented by a graph $G = (V, E)$, where V denotes the set of nodes (representing hosts and 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 model link properties with three functions defined on the edges of the graph. Specifically, we assume that each link has a *cost*, modeled by a cost function $P : E \to R^+$, where R^+ denotes the set of positive real numbers. If we are only interested in counting the number of edges in distribution trees, we will use the unity function for P, however more complex cost functions are accommodated by our algorithms. In the static case, $P(e)$ will represent a time-independent cost metric; in the dynamic case, $P(e)$ will represent a cost per unit time for link usage.

We assume each link has a *bandwidth* value, modeled by a bandwidth function $B : E \to R^+$. This represents bandwidth available on the link for this multicast service. This assumption, while fairly conventional for theoretic studies of networking problems, deserves some additional comment. The availability of a known and stable available bandwidth is reasonably consistent with a QoS-based network that makes some form of resource reservations. Indeed, fixed rate sources will have a difficult time operating in an environment that cannot ensure the availability of the required bandwidth. For a pure best-effort network, such a value might represent an estimate of medium to long-term average available bandwidth, suitable for planning the initial trees used by rate-adaptive sources. If and when the bandwidth deviates from the estimate, a rate-adaptive source can modify its transmission rate.

We consider a single service delivered by a set of n servers $S = \{s_1, \cdots, s_n\}$ to a set of clients C. While a client 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*. When we assume the set of clients is static over the duration of the multicast content distribution session, the set C is fixed. When we assume the set of clients are dynamic, the set C changes over time. More details regarding the model for dynamic behavior are given in Section IV when we develop the dynamic case.

Allocation of clients to servers will result in the establishment of n multicast trees[1], $T_i = (V_i, E_i)$, $V_i \subseteq V, E_i \subseteq E$, such that $s_i \in V_i$ and all clients c_j that select server s_i satisfy $c_j \in V_i$. The set $\mathcal{T} = \{T_i;\ i = 1, \ldots, n\}$ completely specifies the state of the system.

In the rate-adaptive case, we will use r_i to denote (in different contexts) the transmission rate of server i or the reception rate of client i. We will make clear whether we are referring to server or client rate. We will also model the fact that some applications may have a non-zero minimum transmission rate. We use δ to denote the (application-specific) minimum rate required. The specification of a non-zero value for δ means that some clients may be unable to receive service from one or more servers, depending on available link bandwidth.

For rate-adaptive servers, we will be interested in two metrics for the collective performance of the clients. One is the *total receiving rate* of all clients, i.e., $\sum_i r_i$, where r_i is the receiving rate of client i. The other is the *total inter-receiver fairness* [17]. Informally, inter-receiver fairness measures the deviation from the ideal rate that each client receives, where "ideal" is defined as the rate a client would receive if unconstrained by any other clients. One specific definition for total inter-receiver fairness is $\sum_i r_i / r_i^*$, where r_i is the receiving rate of client i and r_i^* is the ideal rate. The reader is referred to [17] for more details on inter-receiver fairness functions.

[1]It is straightforward to generalize all of our results to the case in which a single rate-adaptive server has multiple streams, each serving a different set of clients at a different rate [18]. We omit the generalization here for ease of exposition.

III. Static Multicast Server Selection

A. Problems and Complexity

We define three basic problems, one for the fixed rate case and two for the rate-adaptive case.

For the case in which the servers operate at a fixed rate, a *feasible set of trees* are those in which all links support the fixed rate. That is, for fixed rate r, we are interested in the subgraph of G containing edges $e \in E$ such that $B(e) \geq r$. Tree edges must all come from this subgraph. Amongst feasible sets of trees, we can then identify two basic allocation problems, one based on minimizing cost, and one based on minimizing delay.

The *Minimum Cost Allocation Problem* is to find n feasible multicast trees such that the total cost of tree links is minimum, i.e., minimize $\sum_{i=1}^{n} \sum_{e \in E_i} P(e)$. This is a generalization of the Steiner tree problem [19], and hence is NP-hard.

It turns out that we can transform the (multiple-tree) Minimum Cost Allocation Problem to a single tree Steiner problem, thus allowing us us to use any of the Steiner tree heuristics to solve it [20]. The transformation method is as follows:

1. 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 clients $C = \{c_1, \cdots, c_m\}$. This is a minimum Steiner tree problem for G' with a set of compulsory nodes $\{s_0\} \cup C$.
3. Let T' denote the Steiner Tree. Remove any edge (s_0, s_i) $(1 \leq i \leq n)$ if it is in T'. We get a forest $\mathcal{T}$, which is the final allocation of clients to servers.

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

When the servers are *rate-adaptive*, the rate of a server will depend on its tree configuration and the *rate adaptation policy*, which determines the rate for a server given the relevant operating conditions. In particular, we assume that the sending rate for a server is determined by the *minimum share* of bandwidth on all links in the server's distribution tree[2]. Because multiple trees may share a given link, a server's share of bandwidth depends on a policy for allocating the bandwidth of shared links. We admit any policy that satisfies the constraint that the total bandwidth used by the multicast session does not exceed the available bandwidth on any link. That is, for all links $e \in E$,

$$\sum_{e \in E_i} r_i \leq B(e), \qquad (1)$$

where r_i is the rate of the server rooted at tree T_i.

We define two optimization problems for rate-adaptive servers.

The *Maximum Rate Problem* is to find n multicast trees and a rate sharing policy, such that the total rate to clients is maximum, i.e., maximize $\sum_{i=1}^{m} r_i$, where r_i is the reception rate of client i.

The *Maximum Fairness Problem* is to find n multicast trees and a rate sharing policy, such that the total fairness to clients is maximum, i.e., maximize $\sum_{i=1}^{m} r_i / r_i^*$, where r_i is the reception rate of client i and r_i^* is the ideal rate of client i.

Both problems are NP-hard, as can be shown via a transformation from the 3-SAT problem [21]. We state and prove the result for the Maximum Rate Problem; it is straightforward to show it holds for the Maximum Fairness Problem as well.

Theorem. Given a network $G = (V, E)$, a bandwidth function B defined on E, n multicast servers $s_1, \cdots, s_n \in V$, m clients $c_1, \cdots, c_m \in V$, the following problem is NP-hard: Is there an allocation of clients to servers with server i operating at rate r_i such that the total rate of all the clients is greater or equal to R, subject to the link capacity constraint in equation (1)?

Proof. The details of the proof are in Appendix A. The main idea of the transformation from the 3-SAT problem is to use one server per literal and one client per clause, with edges in the graph to express the presence of the three literals per clause. The selection of a server by a client corresponds to the satisfaction of the clause represented by the client by the literal represented by the server. Auxiliary edges and vertices are also used, as described in the appendix.

B. A Special Case

The general static multicast server selection problems are NP-hard, thus we are unlikely to be able to find efficient algorithms that give optimal solutions. Because of its correspondence to the Steiner Tree Problem, the Minimum Cost Allocation Problem can use any of the Steiner Tree heuristics. Hence we turn our attention to the problems in the rate-adaptive case.

[2]Other rate adaptation policies are certainly possible. For example, the server might have a discrete set of operating rates and select the largest that does not exceed the minimum share of bandwidth on any link in the tree.

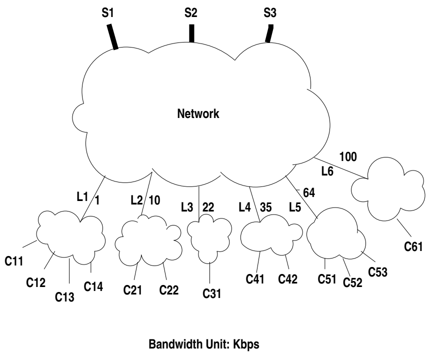

Fig. 1. Example special case

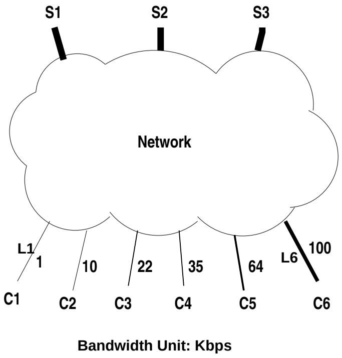

Fig. 2. Example special case, single client groups

Before we explore heuristic solutions to the general problems that arise for rate-adaptive servers, we are interested in finding a simplified version of the rate-adaptive problem that may arise in practical situations and can be solved efficiently.

The special case we focus on is one in which the clients can be clustered into groups such that the bottleneck link for each group occurs at the (single) access link between the group's domain and the rest of the Internet. (See Figure 1.) The implicit assumption is that the links within a client domain and within the backbone of the Internet have sufficient bandwidth so as not to be the bottleneck.

A particular bottleneck link carries only traffic to a single group, and all clients in the group will be served from the same server. Because the bottleneck links are not shared across multicast distribution trees, the problem reduces to selecting a server for each client group. The rate of the server will be the minimum rate of the bottleneck link for any client group that selects that server. If any bottleneck link has availabe rate less than δ, the minimum rate of the application, then those clients will be unable to receive the service.

We describe the solution to this problem for client groups consisting of one client each, as depicted in Figure 2. We then explain the straightforward modifications necessary for larger client groups.

Assume client c_j can receive at rate r_j, which is the bottleneck access link rate. Assume that there are n servers, and the rate of each server is the minimum rate of all the clients selecting it. If we order the clients such that $r_j \le r_{j+1}$ $(1 \le j \le m-1)$, we can formulate the problem as follows: Given m clients with rates r_j in increasing order, select n rates from them as server rates. Assume that the n indices marking the selected rates are k_i $(1 \le i \le n)$. We let $k_1 = 1$ because we assume that each server must adapt to the lowest rate of clients selecting it and every client must select one of the server. In other words, we have a constraint $(r_i \ge r_1)$ on the minimum rate of clients. If we define $k_{n+1} = m+1$, we can let client j $(k_i \le j < k_{i+1})$ select server i and its effective rate is r_{k_i}. Thus the selection problem reduces to picking the n server rates (indices) amongst the m client rates.

B.1 Optimal allocation algorithm

The brute force solution to the problem involves trying every possible way to select $n-1$ server rates from $m-1$ client rates. When $n << m$, the complexity is exponential $O(m^n)$.

We develop an algorithm with complexity $O(nm^2)$, by first transforming it to the shortest path problem in a graph and then using Dijkstra's algorithm to solve it. We present an algorithm for solving the maximal rate problem, and also show how to modify it for the maximal fairness problem.

The optimal total rate for m clients is $\sum_{j=1}^{m} r_j$. Instead of maximizing the total rate of clients directly, our algorithm solves the problem by minimizing the deviation from the above optimal rate.

We represent each possible choice of server rates from amongst client rates using a graph, with the vertices arranged in rows and columns as follows. There is one column of vertices for each server; there is one row of vertices for each client. Vertex $v_{j,i}$ in row j and column i represents the potential for server i to select the rate of client j. Vertices in the same column are arranged in sorted order by client bottleneck rate. Figure 3 illustrates the graph for the example in Figure 2.

The edges in the graph are used to represent the performance associated with the choice of particular rates for two adjacent servers. That is, an edge from $v_{j,i}$ to $v_{k,i+1}$ $(j \le k)$ represents that server i selects the rate of client j while server $i+1$ selects the rate of client k. The weight on this edge is the "deviation" from ideal for clients $j, \cdots, k-1$ that results from this choice. These clients will all be assigned to server i with rate r_j; all except client j will therefore receive at a rate lower than their bottleneck rate.

A solution is represented by a path through this graph that includes one vertex per column, thus selecting one client rate per server. With the edges and weights as described, a shortest path corresponds to an optimal solution.

Formally, we define the *deficit* from i to j $(i \le j)$ as the difference between the bottleneck rates of clients i and j, i.e., $D(i,j) = r_j - r_i$. If client j selects a server sending at rate r_i (note $r_i \le r_j$), the wasted capacity by j is $D(i,j)$. The *cumulative deficit* from i to j (denoted as $D_C(i,j)$)

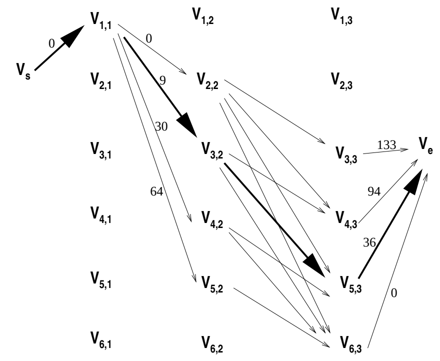

Fig. 3. An example solution

is the sum of the deficits from i to all k $(i \le k \le j)$, or $D_C(i,j) = \sum_{k=i}^{j} D(i,j)$. Define $D_C(i,j) = 0$ if $i \ge j$.

We define a deficit digraph $G_d = (V_d, E_d)$, where $V_d = \{v_{j,i}\} \cup \{v_s, v_e\}$ $(j = 1, \cdots, m, i = 1, \cdots, n)$ and E_d is given as follows. Connect v_s to $v_{1,1}$ with an edge of length 0. For $1 \le j \le k \le m, 1 \le i \le n-1$, connect $v_{j,i}$ to $v_{k,i+1}$ with an edge of length $D_C(j, k-1)$. For $1 \le j \le m$, connect $v_{j,n}$ to v_e with an edge of length $D_C(j,m)$. For the rates of clients in Figure 2, we can generate a graph in Figure 3 (not all edges are shown).

Using Dijkstra's algorithm, we can compute a shortest path from v_s to v_e. The path must be in the form $(v_s, v_{1,1}, v_{k_2,2}, \cdots, v_{k_j,j}, \cdots, v_{k_n,n}, v_e)$. These k_i's $(2 \le i \le n)$ are indices we use to partition the client set. The shortest path in the problem in Figure 3 is $(v_s, v_{1,1}, v_{3,2}, v_{5,3}, v_e)$ (shown as bold lines) and the optimal solution to the problem is to set servers s_1, s_2, s_3 at rates r_1, r_3, r_5, or 1, 22, 64 and allocate clients c_1, c_2 to s_1, clients c_3, c_4 to s_2 and clients c_5, c_6 to s_3.

For the maximum fairness allocation algorithm, we only need to modify the deficit from i to j to be $D(i,j) = 1 - r_i/r_j$. The remainder of the transformation and solution remain as described.

When the groups contain multiple clients, each vertex $v_{j,i}$ in the transformed graph represents the potential for server i to select the bottleneck rate of client group j. All clients in the group will receive at this rate. The deficit from i to j is then $D(i,j) = n_j(r_j - r_i)$, where n_j is the number of clients in group j and r_j is the bottleneck rate for group j.

B.2 Performance results

To illustrate the use of the algorithm described above, consider a system with 400 receivers. Their bottleneck rates are randomly set between 0.1 to 0.9. The number of servers varies from 1 to 15. We compare five algorithms:

1. Maximize Average Rate, as described above.
2. Maximize Fairness, as described above.
3. Random. Each client randomly selects a server, and the rate of server is set to the lowest of the rates of the clients selecting this server.
4. Equally Spaced Server Rates. Assume that the range of rates is pre-determined. Each server is assigned a rate evenly spaced from each other. Each client selects the

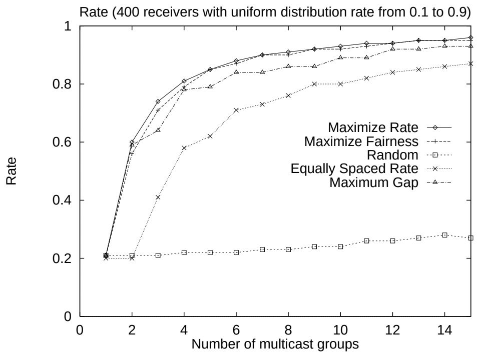

Fig. 4. The average rate of clients

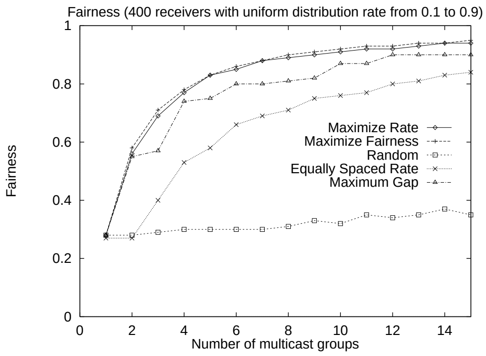

Fig. 5. The inter-receiver fairness

fastest one from the set of servers that have a lower (or equal) rate than this client.

5. Maximum Gap. This is an approximation to the optimization algorithm above. It operates as follows. Assume client rates are in ascending order. Calculate the difference between the rates of two clients in adjacent positions in that order. Divide the clients into groups at the place where the difference is biggest. If there are several locations with the same biggest distance, divide arbitrarily if there are no consecutive pairs that have the same biggest distance. If there are consecutive pairs with the biggest distance, they constitute a chain. We can divide at the middle of the longest chain. For the problem in Figure 2, the Maximum Gap solution is to set servers s_1, s_2, s_3 at rates 1, 64, 100, respectively, and allocate clients c_1, c_2, c_3, c_4 to s_1, client c_5 to s_2 and client c_6 to s_3.

In Figure 4, we compare the average rate of each algorithm. We use the average of bottleneck rates of all clients as a comparison baseline. The rate shown in the plot is the ratio of the average of actual rates of clients over the average of bottleneck rates. As the number of servers increases from 1 to 15, the average rate in all cases increases. The random selection algorithm does not increase much while all other algorithms demonstrate a sharp increase when the number of servers increases from 1 to 6. The maximum rate algorithm has the highest average rate, and the maximum fairness algorithm is very close to it. They both perform better than equally spaced rate and maximum gap algorithms.

Figure 5 shows the fairness of each algorithm. As expected, the maximum fairness algorithm has the largest fairness value, and the fairness of the maximum rate algorithm is very close to it. They both have a larger fairness value than maximum gap and equally spaced rate algorithms. The random selection has a very low fairness value. This is because all servers are operating at rates of some low-end clients, and all the clients are receiving at a very low rate. Therefore, both the average rate and fairness are very low when using random selection.

These results demonstrate that increased sophistication in the selection algorithm has a considerable performance payoff. The presence of rate-adaptive servers requires coordination across clients to avoid the "lowest common denominator effect" illustrated by the random result. The equally spaced rate improves upon the random result, however, the three most sophisticated schemes offer additional improvement varying from 10% to 40%.

C. *The General, Rate-Adaptive, Static Problem*

We turn to the development of heuristics to solve the general static problem for rate-adaptive servers. In doing so, it is useful to divide the problem into two components:

- **Selection,** which chooses a server and path for each client.
- **Rate allocation,** which determines the bandwidth sharing policy when two or more trees share a link.

The first of these components is essentially a routing problem; the second requires coordination across different trees.

C.1 Three Heuristics

We examine the performance of three heuristics, *Shortest Path*, *Widest Path*, and *Optimized Widest Path*. These heuristics differ in their assumptions about the underlying routing and their approach to handling the two components of the problem.

In the *Shortest Path* heuristic, each client selects the nearest server based on the hop count, with a tie-breaking mechanism to ensure consistent choice across different clients if two servers are equi-distant. Consistent choice makes sure that there are no shared links between different multicast trees established this way, hence there is no rate allocation component to the problem. Instead, we set the server rate to be the minimum bandwidth of all the links in the server's tree.

The routing assumptions of this heuristic are reasonably consistent with current proposals for multicast routing that build source-rooted shortest path trees. Implementing the heuristic would, of course, also require a mechanism for a client to discover which source was closest before initiating a join.

In the *Widest Path* heuristic, each client selects a server that has the highest bandwidth from the server to the client. If multiple paths have the same bandwidth, we choose the shortest one.

Unlike the Shortest Path heuristic, the Widest Path heuristic may result in multiple trees sharing a link. Thus, we must also solve the rate allocation component of the problem. We use the following Greedy algorithm, which maximizes the sum of the rates across all clients. That is, the algorithm finds an *optimal* rate allocation, given a particular set of trees.

At each step, this algorithm assigns a rate to the server with the largest number of clients that has not yet had its rate assigned. The chosen rate is as large as possible, subject to the link constraints. That server and clients are then "removed" from the problem, and the link rates are adjusted accordingly. Assume S is the set of all servers. The algorithm is as follows:

Step 1. For each edge e_j, let $b'_j = b_j - \ell_j \delta$, where ℓ_j is the number of servers currently using this edge and δ is the minimum rate of the application. If any b'_j is less than 0, then there is no solution to the original problem and exit; otherwise,

Step 2. Select a server s_i from S with the largest k_i (number of clients). For all the edges e_j that s_i uses, select the smallest b'_j. Set x_i to this value.

Step 3. For all the edges e_j that s_i uses, let $b'_j = b'_j - x_i$. Remove s_i from S. If S is not empty, go to Step 2.

Step 4. For server i, set its rate to be $x_i + \delta$.

The routing assumptions of this heuristic are consistent with some proposals for QoS-based routing that consider bandwidth in making routing decisions. The rate assignment algorithm as described requires that a central location be aware of the set of trees in order to coordinate. Hence, this approach is probably best suited for a link-state QoS-based model for routing.

In the *Optimized Widest Path* heuristic, we combine the tasks of selection and rate assignment. Observe that with the Widest heuristic, a client selects a path that is best for itself. However, the bandwidth of paths connecting to the same server may vary a lot. When a server has clients with considerable diversity in path bandwidth, the Optimized Widest heuristic forces those clients at the low end of path bandwidth to select a different server or form another server. This allows the particular server to operate at higher bandwidth, often with minimal penalty for the clients forced to another server.

Recall that the server set is $S = \{s_1, \cdots, s_n\}$ and client set is $C = \{c_1, \cdots, c_m\}$. Let M denote the maximum bandwidth of any edge in E.

Step 1. Add an extra node s_0 and connect s_0 with each s_i $(i = 1, \cdots, n)$ with an edge of bandwidth $M + 1$.

Step 2. Use Dijkstra's algorithm to set up a tree so that s_0 has maximum rate along this tree to any node in the graph. This consists of the following steps.

2.1) Set selected set $N = \{s_0\}$ and $\bar{N} = V - N$.

2.2) Select an edge $e = (v_i, v_j)$ such that $v_i \in N$ and $v_j \in \bar{N}$ and the residual bandwidth of e is the biggest. (Initially, the residual bandwidth is the available bandwidth $B(e)$ of the edge. As the algorithm progresses, it will modify the residual bandwidth of edges.) Add this edge to the tree.

2.3) Add v_j to N and remove v_j from $\bar{N}$.

2.4) Repeat 2.2) and 2.3) until $N = V \cup \{s_0\}$.

Step 3. From subtrees rooted at each server, pick one with the largest number of clients in the subtree. Optimize this subtree as described in the following steps.

3.1) Prune those non-client leaves (recursively).

3.2) Assume the clients selecting the server are $1, \cdots, w$ and the bottleneck rate to client i is r_i $(1 \leq i \leq w)$. Without loss of generality, we assume $r_i \leq r_{i+1}$ for $1 \leq i \leq w-1$. We determine a k such that the value $(w-k+1)r_k$ is maximized. We will go over a feasibility test and change k to 1 if the test is not successful. We let clients $k, \cdots, w$ stay with this server and form a server, which will operate at rate r_k. Clients $1, \cdots, k-1$ are pruned from the subtree.

3.3) The subtree with clients $k, \cdots, w$ is pruned to remove unnecessary leaves, if any. Reduce the bandwidth of each edge in the pruned subtree by the server rate. These edges get a new residual bandwidth.

3.4) Remove clients in the subtree $(k, \cdots, w)$ from the client set C.

3.5) If the number of servers from this server reaches the limit of maximal number of servers from one server, then remove the server from the server set S.

Step 4. If $C = \emptyset$, stop; else goto step 2.

One remaining step in the algorithm is the feasibility test, in which we generate a maximum rate tree in the same way as in Step 2 for those nodes in current client set C minus those nodes $(k, \cdots, w)$ selected. We can get a rate for each client in this tree. If the rate of each of them is greater than or equal to the minimum rate requirement, the feasibility test is successful; otherwise, the test fails.

This heuristic assumes availability of topology and bandwidth information in a central location, in order to run the algorithm. This assumption is most consistent with link-state, QoS-based routing environments.

C.2 Performance Results

In this subsection, we evaluate the performance of the heuristics. In addition to the Shortest, Widest, and Optimized Widest heuristics, we calculate an upper bound on the rates of clients. Each client can receive at its bottleneck rate if there are no other clients. Usually, this rate cannot be achieved if we have multiple clients in the session simultaneously. This is a loose bound on what we can achieve when the clients are receiving at the same time. We will compare the average rate of clients by the heuristics with this bottleneck average, called the *Isolated Rate* measure.

We generate two kinds of topologies using GT-ITM [22]. One is a random graph with 400 nodes. Another is a transit-stub graph with 400 nodes. We have 1 to 16 servers randomly placed in the graph. We experiment with two kinds of client densities: either 5% of the nodes are clients or 25% of the nodes are clients, also randomly placed in the network. The results we report are averages over a hundred simulations with randomized link bandwidths and client/server placement. This is intended to give a level of confidence in how these heuristics will perform (on average) in realistic settings. We show the results for the transit-stub graph with 25% of nodes being clients. Results in other cases are similar and can be found in [23].

Figure 6 shows the average rate of all clients. As the number of servers increase, the average rate increases in all cases. The Widest heuristic has a higher average rate than

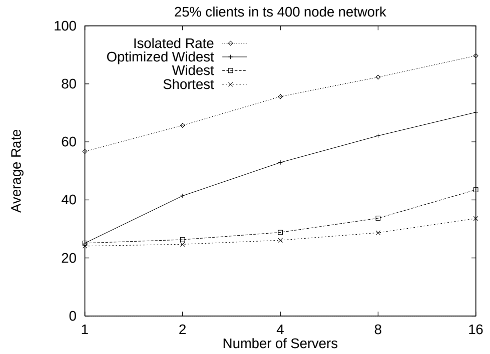

Fig. 6. The average rate

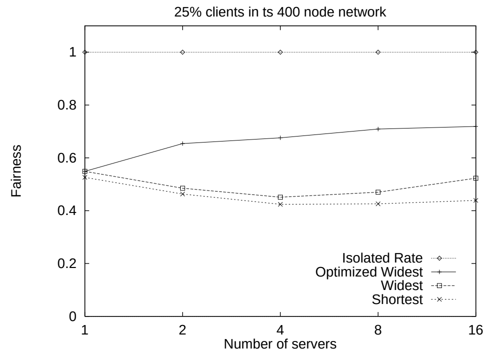

Fig. 7. The inter-receiver fairness

the Shortest heuristic, and the Optimized Widest further improves over the Widest if the number of servers is more than one. The average rate of Optimized Widest is much closer to Isolated Rate than Widest and Shortest.

Figure 7 shows the fairness. From the definition in Section II, it is obvious that the fairness in the Isolated Rate case is always one. The Widest heuristic has a higher fairness value than the Shortest heuristic, and the Optimized Widest improves upon this when the number of servers is greater than one. Note that the fairness in the Shortest and Widest cases decreases when the number of servers increases from one to two, then to four. This is because the bottleneck rate increases faster than the actual rate obtained by these heuristics, thus, the average of ratio of the actual rate over the bottleneck rate decreases. Similar phenomenon is also observed in other cases.

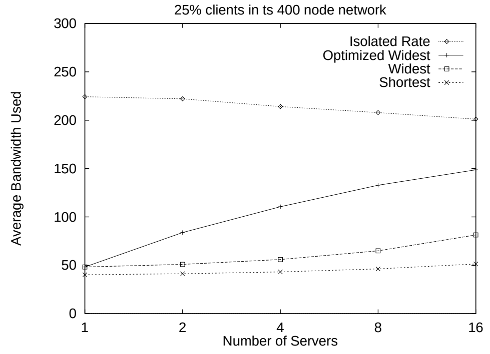

Fig. 8. The bandwidth used

Figure 8 shows the bandwidth used. As expected, Widest uses more bandwidth than the Shortest, and the Optimized Widest uses more bandwidth than the Widest. As the number of servers increases, the bandwidth used by three heuristics also increases. This is because the rate of servers and clients can be higher when we have more servers. We notice that in our simulation, when the number of servers increases beyond a certain point, the bandwidth used will decrease. This is because in those situations, the average rate cannot increase much, but the distance from a server to a client becomes shorter. Therefore, the bandwidth used decreases because the number of hops from a server to its clients is smaller. The bandwidth used by Isolated Rate case decreases when the number of servers increases. This is because it may find a shorter path when there are more servers.

IV. Dynamic Multicast Server Selection: Fixed-Rate Servers

In many multicast applications, the membership of clients changes dynamically. A client 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 (e.g., [24], [25], [26]). 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 clients 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.

In addition to the basic modeling assumptions of Section II, we assume there are m *potential clients* $C = \{c_1, \cdots, c_m\}$. We assume that each client remains a member of the multicast session for a random period of time (the *time-in-session*) with 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 $1/\lambda$.

In this context, the cost function P defined earlier is the cost *per unit time* of using this edge as a part of a multicast tree. Another cost function $H : E \to R^+$ is defined on the the edges of G. This represents the cost of pruning or grafting this edge onto a multicast tree.

At any given time, the *client state* can be described as $J = \langle j_1, \cdots, j_m \rangle$, where $0 \le j_k \le n$ for $1 \le k \le m$. If $j_k = 0$, the client c_k is not in the multicast session at this time. If $j_k > 0$, the client c_k is in the multicast tree rooted at the server s_{j_k}. A transition of state occurs when a client joins or leaves the session.

A. The Minimum Cost Problem

For the case in which the servers operate at a fixed rate, the dynamic changes to trees must preserve the feasibility of those trees. That is, in each change to the set of trees, all links must support the fixed rate. Given the restriction that the trees always remain feasible, we can define a problem to minimize the two types of costs involved in maintaining dynamic trees:

- **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 (defined by the function P) of the edges in the trees being used. More formally the state occupancy cost in state $\mathcal{T}$, which contains the trees $T_i = (V_i, E_i)$, $i = 1, \cdots, n$, is given by $\mathrm{OCost}(\mathcal{T}, t) = t \sum_{i=1}^{n} \sum_{e \in E_i} P(e)$, 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, as a result of a client joining or leaving. It is a function of the costs (defined by the function 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 $T_{i1} = (V_{i1}, E_{i1})$ to another set of induced trees $T_{i2} = (V_{i2}, E_{i2})$, $i = 1, \cdots, n$. 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, and $t = \sum_{i=0}^{r} t_i$. The expected total cost of running the system for the t time units is given by

$$\mathrm{TotCost}(t) = E[\alpha \sum_{i=0}^{r} \mathrm{OCost}(\mathcal{T}_i, t_i) + (1-\alpha) \sum_{i=0}^{r-1} \mathrm{TCost}(\mathcal{T}_i, \mathcal{T}_{i+1})] \quad (2)$$

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

B. Policy-based Solutions

The dynamic problem inherently requires an on-line algorithm to decide what to do each time the client set changes. If the transition from some client state $J = \langle j_1, \cdots, j_m \rangle$ to another client state $J' = \langle j'_1, \cdots, j'_m \rangle$ is triggered by client 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 client ℓ leaving the session, then we know $j_\ell > 0$ and $j'_\ell = 0$. For any other client, 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 client will not change other clients' 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 client can change other clients' 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.

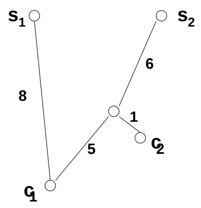

Fig. 9. The network setup

In each state, a *decision* specifies a next state for each possible join/leave event. Assume the current client state is $J = \langle j_1, \cdots, j_m \rangle$. If $j_\ell = 0$, the client ℓ may join a group. A decision in this case would indicate the server to which this new client should be allocated. In addition if our system allows switching, a decision would include any changes to the allocation of other clients as a result of this particular join event occurring. If $j_\ell > 0$, the client 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 clients. 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 $\liminf_{t \to \infty} \frac{\mathrm{TotCost}(t)}{t}$.

C. An Optimal Solution to the Minimum Cost Problem

If client 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 [27] 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 clients 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.

C.1 Tradeoff Between State Cost and Transition Cost

We illustrate the interesting properties of the optimal policy using the simple network setup shown in Figure 9. There are two servers, s_1 and s_2, and two clients, 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 assume the average arrival rate and departure rate are 0.1. We can

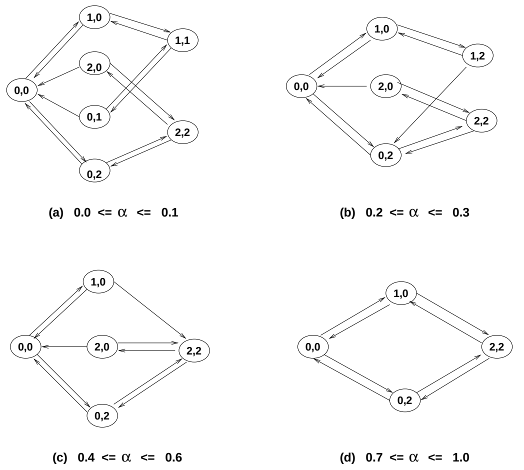

Fig. 10. The tradeoff between state occupancy cost and transition cost

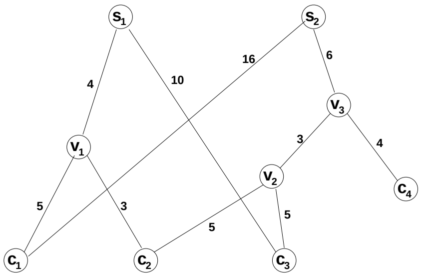

Fig. 11. A server selection problem with 2 servers and 4 clients

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 10, 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 client. For example, $\langle 2, 1\rangle$ indicates that client c_1 selects server s_2 and client c_2 selects server s_1. A value of 0 means that the client is not in the multicast session. In the figure, we show only those states reachable from state $\langle 0, 0\rangle$ by the policy. In all cases, client c_1 joins s_1 if neither c_1 nor c_2 is in the session, i.e., state $\langle 0, 0\rangle$ to state $\langle 1, 0\rangle$. Similarly client c_2 joins s_2, i.e., state $\langle 0, 0\rangle$ to state $\langle 0, 2\rangle$. The more interesting behavior occurs when the number of clients in the session goes from 1 to 2 (due to a join) or from 2 to 1 (due to a leave). We show four cases based on different α values. When α changes from 0 to 1.0, the best policy evolves from minimizing transition cost to minimizing state cost.

C.2 A Larger Example

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

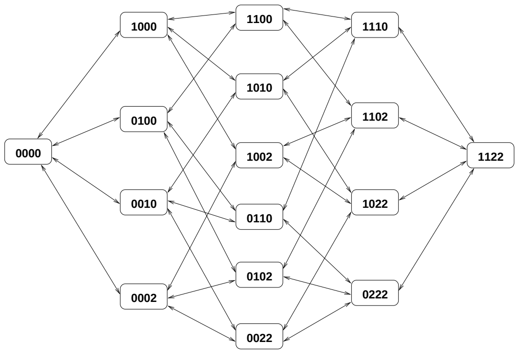

Fig. 12. The solution to the server selection problem with 2 servers and 4 clients

In Figure 11 we have two servers s_1, s_2, and four clients c_1, c_2, c_3, c_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 average arrival rate is $\lambda = 10^{-4}$ and the departure rate is $\mu = 10^{-3}$ for all the clients. 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 12, where states are in the form of tuples as previously defined.

The join and leave behavior of the clients is as follows:

- **Join Behavior**

Client c_1 always selects s_1 and client c_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 clients c_2 and c_3.

Consider the state transition from $\langle 0002\rangle$ to $\langle 0022\rangle$. client c_3 will select s_2 if c_4 is already in the tree because the extra cost is 8 (from c_3 to v_3), which is smaller than selecting s_1 with an extra cost of 10 (from c_3 to s_1). Client c_3 will select s_1 if c_4 is not in the tree (e.g., the transition from $\langle 0000\rangle$ to $\langle 0010\rangle$) because the cost of selecting server s_1 is smaller than the cost of selecting server s_2. Similar is the case for client c_2. If c_3 is connected to server s_2, client c_2 will select server s_2 (e.g., the transition from $\langle 0022\rangle$ to $\langle 0222\rangle$), otherwise it will select server s_1 (e.g., the transition from $\langle 0000\rangle$ to $\langle 0100\rangle$).

An important assumption in our model is that we only consider the shortest path from a server to a client. So the candidate nodes that a client 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 client will select the nearest in-tree node in its shortest paths to all servers and establish a path from that node to the client.*

- **Leave Behavior**

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

Consider state $\langle 0022\rangle$, where clients c_3 and c_4 are both connected to s_2. When client c_4 leaves the tree, c_3 changes from selecting server s_2 to s_1, leading to state $\langle 0010\rangle$. This is because c_3 can share links with c_4 in the original state, but the links are not shared once c_4 leaves.

Now we analyze state $\langle 0222 \rangle$, where clients c_2, c_3 and c_4 are connected to server s_2. When c_3 leaves, client c_2 changes from s_2 to s_1. We observe that c_2 shares a link with c_3. This indicates that when a client leaves, we should make an adjustment to those clients sharing links with the departing client. We can reduce the cost by letting them re-select a server.

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

D. Heuristics for the Minimum Cost Problem

Our experience with the MDP-based solution leads us to some possible selection algorithms which we describe here.

D.1 Join Algorithms

Nearest Server Algorithm. In the nearest server algorithm, a client 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 client. The problem with this approach is that it will 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 client. Therefore, the cost of the tree may be higher.

Nearest Tree Algorithm. In the nearest tree algorithm, a client joins the multicast group whose routing tree requires the shortest path to be grafted in order to join the group. This leads to a lower cost tree, but it is hard to implement. Usually a joining client 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 IV-C.2, we propose this new selection algorithm. Instead of letting a client select the nearest server or the nearest tree, we let the client select the nearest node among those that are in the shortest paths from servers to the client and already in the multicast trees.

This algorithm is simpler than the nearest tree algorithm if we do not know what nodes are already in the multicast trees, because we need consider only those nodes in the shortest paths.

D.2 Leave Algorithms

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

With Adjustment. If we allow switching, we may be able to reduce the tree cost. The idea is to let all the clients sharing edges with the deleted node re-select a server using a join algorithm. If the client is not a leaf, all the clients in the subtree rooted at this node will re-join a group (re-select a server) using previous join algorithms. If the client 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 client). We let each client in the subtree rooted at v re-select a server.

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

In the nearest server algorithm, the client's choice is independent of the state of other clients, 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.

E. 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.

E.1 Experimental Settings

Network Topologies. We use the GT Internetwork Topology Models simulator [22] 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 clients, we can control the average number of clients 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 client randomly selects a server. A tree is constructed by the shortest path tree algorithm for each server.

2) Nearest server (SPT).

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. The function value of an edge is used as both the cost rate and the graft/prune cost.

E.2 Numerical Results

In the first experiment we have 2 to 16 servers in the network and the percentage of clients in the multicast session changes from 5% to 30%. When a client joins or leaves the multicast session, we construct a set of trees from scratch. The algorithm we use is based on the KMB algorithm [20] and generated by using the method described in Section III-A 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 13. The x-axis is the ratio of

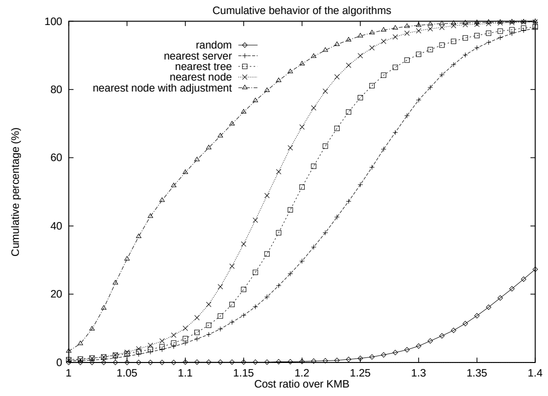

Fig. 13. Cumulative distribution of the cost ratios

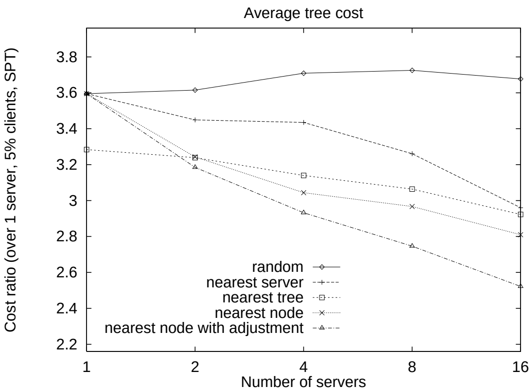

Fig. 15. The average tree cost (30% Clients)

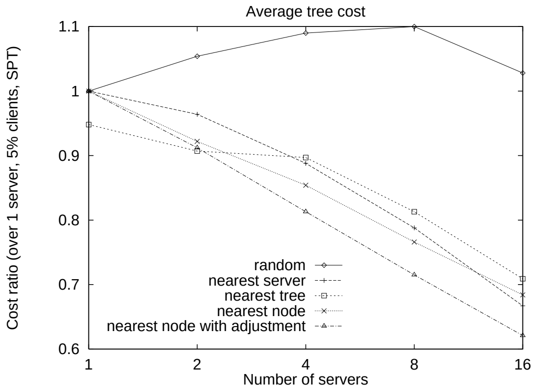

Fig. 14. The average tree cost (5% Clients)

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 the x-value equals 1.4, which means, only 27% of the trees are 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 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 are 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 are within this range.

Figure 14 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 one 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 cost is larger than the nearest node algorithms. When the number of servers increases to four 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 one. This is because the algorithm does not consider sharing between edges and may select a longer path that is not shared with any other client. Note that the 16 server case is better than 8 server case because the effect of increasing the number of servers outweighs the effect of random selection, but it is still worse than the one 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 is more than one server and are better than the nearest tree algorithm when there are four 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 path 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 15 shows the case when the average number of clients in the session is maintained 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 one 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. We also compare how the average tree cost changes with the number of servers. When there are 30% clients, the average tree cost does not decrease as quickly as in the 5% client case. This is because in the 30% client case, there are already a lot of sharing of edges between clients even with 1 server. Increasing the number of servers does not improve the performance as much as in the 5% client case. 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 clients.

Figure 16 shows the join and leave cost when the average

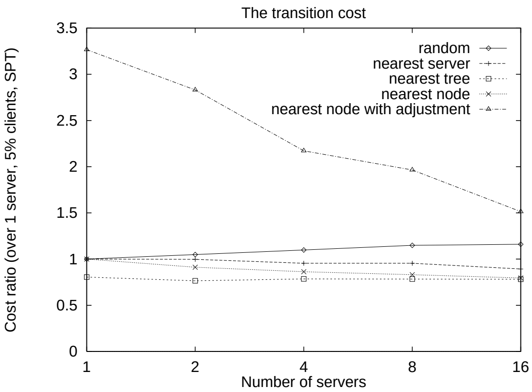

Fig. 16. The join/leave cost (5% Clients)

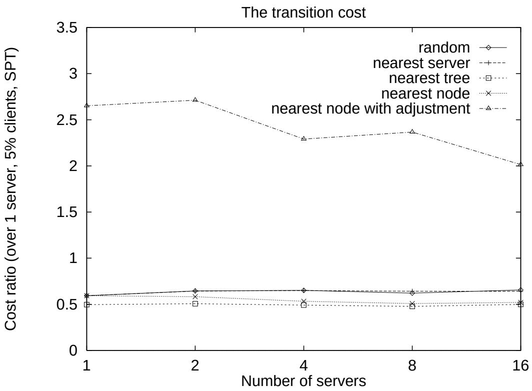

Fig. 17. The join/leave cost (30% Clients)

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 one 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 17 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 for four 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.

V. Dynamic Multicast Server Selection: Rate-Adaptive Servers

Recall that we now have a bandwidth associated with each link in the network, and that our objective is to allocate the clients to the servers so as to either maximize the received bandwidth or to maximize inter-receiver fairness (see Section II).

A. Heuristics for Maximizing Bandwidth

When the servers are rate-adaptive, the join and leave behavior of clients can cause a change in the rate of one or more servers. Recall that we now have a bandwidth associated with each link in the network, and that our objective is to allocate the clients to the servers so as to either maximize the received bandwidth or to maximize inter-receiver fairness (see Section II). We combine what we learned about minimizing the cost of dynamic trees with lessons from the static bandwidth maximization problem to come up with three join heuristics for the maximum bandwidth problem:

1. *Shortest:* Select an in-tree node that is closest to this node based on the hop count.
2. *Widest:* Select an in-tree node that has the highest bandwidth path to the joining node. If there are multiple paths with the same bandwidth, we choose the shortest one.
3. *Greedy*: First, observe that each in-tree node v_k may be in more than one multicast tree. For each tree i, it has an associated rate r_i. From the node wanting to join the multicast session, we can get a widest path to each in-tree node v_k. Assuming the bandwidth of this path is b_k, we know the rate of the server will be $min(r_i, b_k)$ if this node joins.

Based on the above observation, the *Greedy* heuristic will let a client join a multicast group with the biggest benefit value. We define the benefit value to be r_i if $b_k \geq r_i$ and $b_k - n_i(r_i - b_k)$ if $b_k < r_i$, where n_i is the number of clients already in the multicast group for this tree. By doing so, we can make the largest increase from the existing rate.

B. Simulation Results

For the dynamic problem, we compare the performance of three heuristics, Shortest, Widest, and Greedy. As a comparison baseline, we calculate the performance measures with the Optimized-Widest heuristic described in the static case. It reallocates clients to servers from scratch whenever a client joins or leaves. This frequent tree reconfiguration has high overhead and potentially disturbs existing clients, hence it is not particularly practical.

Two kinds of topologies, random and transit-stub, are generated by using GT-ITM. There are 400 nodes in each graph, among which 1 to 16 nodes are servers. The time of a client in the multicast session is exponentially distributed and so is the time outside. By adjusting the average time in and out of the multicast session of clients, we can control the average number of clients in the multicast session to be either 5% or 25% of nodes in the graph. Again we only show the results for the transit-stub graph with 25% nodes

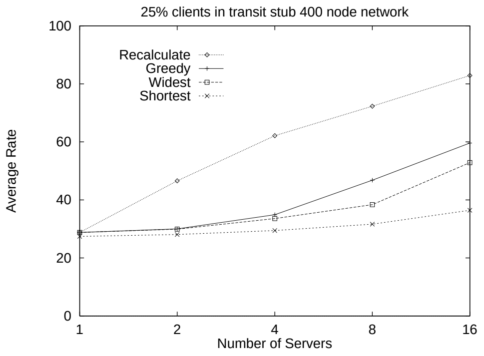

Fig. 18. The average rate of clients

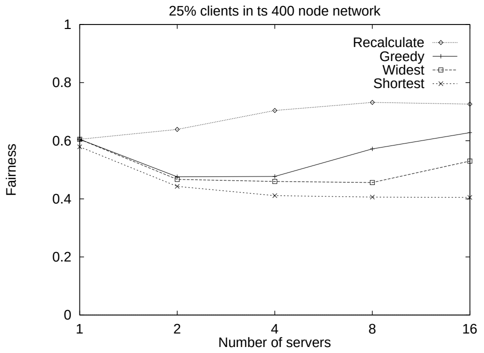

Fig. 19. The inter-receiver fairness

being clients. Interested readers can find similar results in other cases in [23].

Figure 18 to Figure 21 compare the results when we allow only one stream from each server. Figure 18 shows the average rate of clients. We find that the Widest has a higher average rate than the Shortest heuristic and our proposed Greedy heuristic improves over Widest. The Recalculate method can adjust the tree structure when a client joins or leaves, so it can get an even higher average rate.

The fairness in Figure 19 and the bandwidth used in Figure 20 have the same relative ranking as the average rate. The fairness in the Shortest, Widest and Greedy cases decreases when the number increases from one to two, then to four. The reason is that the rate generated by the heuristics does not increase as fast as the bottleneck rate.

Figure 21 shows the join/leave cost. We calculate the hop count of grafting and pruning trees and get an average value over time in each case. The Widest and Greedy heuristics have a higher join/leave cost than the Shortest heuristic because they may choose a longer path with a higher bandwidth. In the Recalculate case, when the number increases from one until eight, the join/leave cost increases. This is because the clients may be allocated to a farther away server. When the number increases from eight, the join/leave cost decreases because it is possible to allocate clients to a closer server, considering we have more servers. One point worth mentioning is that while the Recalculate method can get a much higher average rate and fairness, it has a very high join/leave cost, compared to other heuristics. So it may be infeasible to use the Recalculate method in the dynamic case. Among the three heuristics, our proposed Greedy heuristic has much better performance, while the cost is close to others.

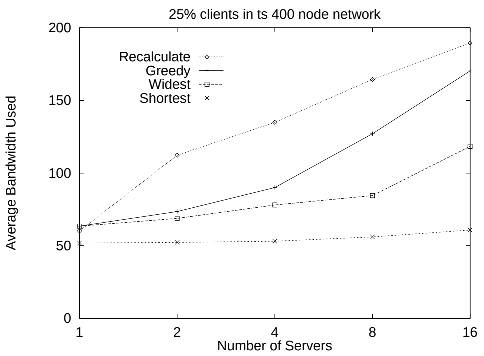

Fig. 20. The bandwidth used

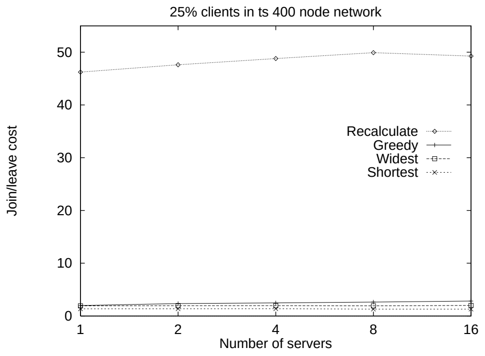

Fig. 21. The join/leave cost

VI. Discussion

We have investigated a collection of problems that arise in the context of multicast server selection. Specifically, we have defined problems in the context of both static and dynamic client groups, and both fixed and rate-adaptive servers. Our contributions include the following:

- Formulation of fundamental problems that arise when multicast servers are replicated. To our knowledge, this is the first work to consider these problems.
- Proofs that the general static problems are NP-complete, including an interesting transformation from the 3SAT problem to the rate-adaptive server selection problem.
- Development of a special case for the static problem that can be solved optimally using a transformation to a shortest path problem. One can argue that this special case (of plentiful bandwidth in the core of the network and within domains, but limited bandwidth in access links) is a reasonable model of current and emerging networks.
- Formulation of dynamic client, fixed-rate server selection as a decision process. Further assumptions allow us to formulate it as a Markov Decision Process and use Howard's policy-iteration algorithm to produce optimal policies for small problem instances leading to an understanding of the properties of the such policies.
- Development of heuristics to solve the general cases of the problems, and investigation of the performance of the

heuristics using simulations.

In the rate-adaptive case, we have assumed that the server can adapt to an arbitrary rate (though an application-specific minimum can be specified). Two obvious alternatives are worth discussing. First, for some applications the server will have a set of discrete operating rates. In this case, a reasonable rate adaptation policy is for the server to operate at the largest discrete rate that does not exceed the bandwidth available on any link in the distribution tree. Some of the algorithms should be modified in this case. For example, in the special case of Section III-B the server rates must be chosen from the discrete rates available, rather than selecting from the bottleneck rates. If the number of servers is smaller than the number of discrete rates, there is still an issue of which rates to pick. This new problem can be solved by a similar transformation to a shortest path problem. Modifications to the heuristics are less immediate, as they may involve issues of "packing" (i.e., looking for paths that satisfy the discrete rate with as little excess as possible).

The second alternative concerns servers that deliver *layered* content. A server that transmits layered content is able to meet the needs of clients with diverse bandwidth capabilities. The server transmits multiple layers, typically each on a different multicast group. Clients that can handle higher bandwidth receive more layers, while clients that have constrained bandwidth receive fewer layers. One can imagine the replication of servers that deliver layered content, thus leading to a selection problem in which each client selects one server per layer, for as many layers as can be delivered. Approaches to solving this problem, including using our work as a building block, are a topic of future work.

The focus of this paper has been on the fundamental problems in multicast server selection. With this groundwork completed, we are now working on the development of practical and deployable protocols that can be used by clients for server selection. Challenges in this area include scalable collection of the information needed to make server selection decisions, and inter-operability with existing multicast routing protocols.

References

[1] Christophe Diot, Walid Dabbous, and Jon Crowcroft, "Multipoint communication: A survey of protocols, functions and mechanisms," *IEEE JSAC*, vol. 15, April 1997.

[2] Pablo Rodriguez and Ernst Biersack, "Continuous multicast push of web documents over the Internet," *IEEE Network Magazine*, vol. 12, no. 2, pp. 18–31, Mar-Apr 1998.

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

[4] Sally Floyd, Van Jacobson, Ching-Gung Liu, Steven Mccanne, and Lixia Zhang, "A reliable multicast framework for lightweight sessions and application level framing," *IEEE/ACM Transactions on Networking*, Nov 1996.

[5] Xue Li, Mostafa Ammar, and Sanjoy Paul, "Video multicast over the Internet," *IEEE Network Magazine*, April/May 1999.

[6] Andy Myers, Peter Dinda, and Hui Zhang, "Performance characteristics of mirror servers on the Internet," in *Proceedings of INFOCOM'99*, Mar. 1999, New York.

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

[8] Robert L. Carter and Mark E. Crovella, "Server selection using dynamic path characterization in wide-area networks," in *Proceedings of INFOCOM'97*, 1997, Kobe, Japan.

[9] Ellen Zegura, Mostafa Ammar, Zongming Fei, and Samrat Bhattacharjee, "Application-layer anycasting: A server selection architecture and use in a replicated web service," *IEEE/ACM Transactions on Networking*, vol. 8, August 2000.

[10] Fang Hao, Ellen W. Zegura, and Mostafa H. Ammar, "Supporting server selection in differentiated service networks," in *Proceedings of IEEE Infocom'2001*, April 2001, Anchorage, Alaska.

[11] Dina Katabi and John Wroclawski, "A framework for global IP-anycast (GIA)," in *Proceedings of ACM SIGCOMM 2000*, 2000.

[12] Mostafa H. Ammar and Li-Ran Wu, "Improving the throughput of point-to-multipoint ARQ protocols through destination set splitting," in *Proceedings of INFOCOM'92*, May 1992, pp. 262–71.

[13] Supratik Bhattacharyya, James Kurose, Don Towsley, and Ramesh Nagarajan, "Efficient rate-controlled bulk data transfer using multiple multicast groups," in *Proceedings of IEEE Infocom'98*, March 1998, San Francisco, CA.

[14] Sneha Kasera, Gisli Hjalmtysson, Don Towsley, and James Kurose, "Scalable reliable multicast using multiple multicast channels," *IEEE/ACM Transactions on Networking*, vol. 8, June 2000.

[15] 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.

[16] S. Y. Cheung, Mostafa H. Ammar, and Xue Li, "On the use of destination set grouping to improve fairness in multicast video distribution," in *Proceedings of INFOCOM'96*, March 1996, San Francisco, CA.

[17] Tianji Jiang, Mostafa H. Ammar, and Ellen W. Zegura, "Inter-receiver fairness: A novel performance measure for multicast ABR sessions," in *Proceedings of ACM SIGMETRICS 98/Performance 98*, June 1998, Madison, Wisconsin.

[18] Zongming Fei, Mostafa Ammar, and Ellen Zegura, "Optimizing the allocation of clients to replicated rate-adaptive multicast servers," in *Proceedings of SPIE International Symposium on Scalability and Traffic Control in IP Networks*, August 2001, Denver, Colorado.

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

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

[21] Michael R. Garey and David S. Johnson, Eds., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.

[22] Ken Calvert, Matt Doar, and Ellen W. Zegura, "Modeling Internet topology," *IEEE Communications Magazine*, June 1997.

[23] Zongming Fei, Mostafa H. Ammar, and Ellen W. Zegura, "Selection of replicated adaptive multicast servers," Tech. Rep. GIT-CC-00-18, Georgia Tech, 2000.

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

[25] Bernard M. Waxman, "Routing of multipoint connections," *IEEE Journal of Selected Areas in Communications*, vol. 6, no. 9, pp. 1617–22, December 1988.

[26] Matthew Doar and Ian Leslie, "How bad is naive multicast routing?," in *Proceedings of INFOCOM'93*, 1993.

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

Appendix

I. NP-hardness Proof of the Allocation Problem

Theorem. Given a network $G = (V, E)$, a bandwidth function B defined on E, n multicast servers $s_1, \cdots, s_n \in V$, m clients $c_1, \cdots, c_m \in V$, the following problem is NP-hard: Is there an allocation of clients to servers with server i operating at rate r_i such that the total rate of all the clients is greater or equal to R, subject to the link capacity constraint in equation (1)?

Proof. The 3-SAT problem is a well-known NP-hard problem. We show that the 3-SAT problem can be transformed to the allocation problem.

Here is the 3-SAT problem: We are given a set of Boolean variables, $X_1, \cdots, X_n$. A literal is a variable X_i or its complement $\bar{X}_i$. Also given is a set of clauses C_j, where each clause contains three literals from three distinct variables. The problem is: Can each variable be set to True or False so that all clauses are true?

The transformation is as follows. Given any instance of 3-SAT problem, with n variables, and m clauses, construct a graph $G = (V, E)$ as shown in Figure 22. Set M to be some big number (like e^{m+n}). The node set V contains X_i and $\bar{X}_i$ $(i = 1, \cdots, n)$, C_j $(j = 1, \cdots, m)$ and auxiliary nodes $A_i, \bar{A}_i, Y_i, U_i, W_i$ $(i = 1, \cdots, n)$ and K_j $(j = 1, \cdots, m)$. For each variable X_i, we have an edge from X_i to A_i, an edge from A_i to Y_i, an edge from $\bar{X}_i$ to $\bar{A}_i$, and an edge from $\bar{A}_i$ to Y_i. Each has bandwidth M. Add an edge from Y_i to U_i with bandwidth 1 and an edge from Y_i to W_i with bandwidth M. For each clause C_j, add an edge from K_j to C_j with bandwidth M. If C_j contains literal X_i, then add an edge from A_i to K_j; if C_j contains literal $\bar{X}_i$, then add an edge from $\bar{A}_i$ to K_j. In the figure, we show a clause C_2 contains $\bar{X}_1$, X_3, $\bar{X}_4$. The servers are X_i and $\bar{X}_i$. The clients are U_i, W_i and C_j. The problem is to decide whether we can find an allocation with total rate $(n+m)M + n$. It is clear that the time to compute the transformation is polynomial.

Assume we can find an allocation with total rate $(n+m)M + n$. Because of constraints imposed by the bandwidth of the edge coming to them, the maximal possible rate of U_i is 1, the maximal possible rate of W_i and C_j is M. To achieve the total maximal rate of $(n+m)M + n$, they must all operate at the maximal rate. For any i, U_i and W_i can only receive from servers X_i and $\bar{X}_i$, so either X_i or $\bar{X}_i$ must have a stream with rate M and another have a stream with rate 1. These two streams cannot come from the same literal (X_i or $\bar{X}_i$) because the bandwidth from X_i to A_i and from $\bar{X}_i$ to $\bar{A}_i$ can only be M.

If X_i has a stream of rate M and $\bar{X}_i$ has a stream of rate 1, we let X_i be True. If X_i has a stream of rate 1 and $\bar{X}_i$ has a stream of rate M, we let X_i be False. Therefore, all literals having a stream of rate M are True. The fact that each C_j must have rate M implies that at least one of literal in C_j must have a stream of rate M. Therefore, every clause must be True.

Conversely, we assume that we can set variables to True or False, so that all the clauses are True. Let U_i select a server (from X_i and $\bar{X}_i$) that is False and W_i select the other one that is True. For any clause C_j, one of its literals must be True because the clause is True. Let C_j selects one of its literals that are True. By verifying against condition (1), we can see following rate allocation is feasible. If X_i is True, we set the rate of X_i to be M and the rate of $\bar{X}_i$ to 1. If X_i is False, we set the rate of X_i to 1 and the rate of $\bar{X}_i$ to M. So U_i has rate 1, W_i has rate M and C_j has rate M. So the total rate is $n + Mn + Mm = (n+m)M + n$. This completes the proof.

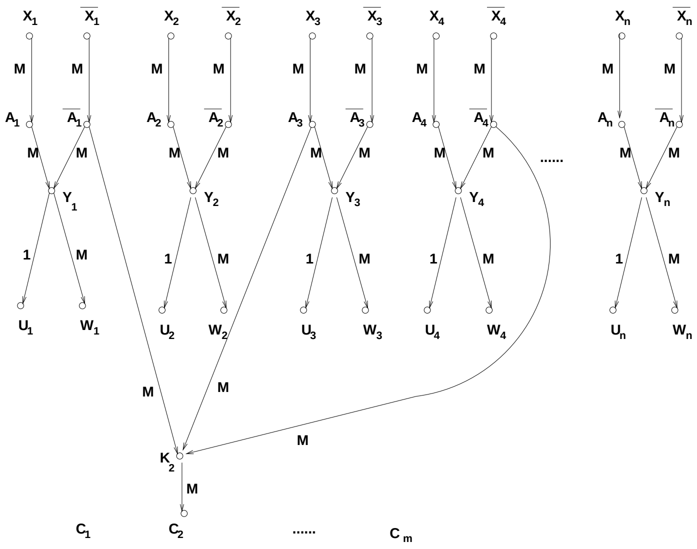

Fig. 22. NP-hard proof of multicast server selection problem