SRC Technical Note
1997 - 012
June 24, 1997

Online Throughput-Competitive Algorithm for Multicast Routing and Admission Control

Ashish Goel, Monika Rauch Henzinger and Serge Plotkin

Systems Research Center
130 Lytton Avenue
Palo Alto, California 94301
http://www.research.digital.com/SRC/

Ashish Goel and Serge Plotkin are at Stanford University. Their electronic mail addresses are: {agoel, plotkin}@cs.stanford.edu.

Abstract

We present the first polylog-competitive online algorithm for the general multicast problem in the throughput model. The ratio of the number of requests accepted by the optimum offline algorithm to the expected number of requests accepted by our algorithm is $O(\log \mathcal{M}(\log n + \log\log \mathcal{M}) \log n)$, where $\mathcal{M}$ is the number of multicast groups and n is the number of nodes in the graph. We show that this is close to optimum by presenting an $\Omega(\log n \log \mathcal{M})$ lower bound on this ratio for any randomized online algorithm against an oblivious adversary, when $\mathcal{M}$ is much larger than the link capacities. We also show that it is impossible to be competitive against an adaptive online adversary.

As in the previous online routing algorithms, our algorithm uses edge-costs when deciding on which is the best path to use. In contrast to the previous competitive algorithms in the throughput model, our cost is not a direct function of the edge load. The new cost definition allows us to decouple the effects of routing and admission decisions of different multicast groups.

1 Introduction

Future high-speed communication networks such as ATM will use bandwidth-reservation in order to achieve Quality of Service (QoS) guarantees. Given a request for a Virtual Circuit (VC), the router has to either accept or reject this request and, if it decides to accept it, allocate the requested bandwidth along a path connecting the endpoints of the VC.

In case of multicast requests, the bandwidth has to be allocated along a tree spanning the nodes participating in the multicast group. In the general case, a multicast request specifies the user (endpoint) and the multicast group that this user wants to participate in. The router should either reject the request or accept it and allocate bandwidth along a path connecting the new endpoint with the already existing tree for this group.

In this paper we present the first polylog-competitive algorithm for the general multicast problem. Our algorithm is randomized since it is impossible to achieve polylog competitive ratio by a deterministic algorithm. The ratio of the number of requests accepted by the optimum offline algorithm to the expected number of requests accepted by our algorithm is $O(\log \mathcal{M}(\log n + \log\log \mathcal{M}) \log n)$, where $\mathcal{M}$ is the number of multicast groups and n is the number of nodes in the graph. If each vertex is allowed to serve at most one multicast group, the competitive ratio simplifies to $O(\log^3 n)$.

Routing and admission control problems in the online setting were extensively studied. In particular, in [3] it was shown how to achieve $O(\log n)$ competitive ratio with respect to throughput for infinite duration requests, i.e. number of accepted requests, and $O(\log nT)$ ratio for requests that specify holding times (durations) upon arrival, where T is the longest duration. The randomized model where the durations are exponentially distributed and the arrivals are Poisson with unknown rates was considered in [9]. They describe a $(1+\epsilon)$-competitive algorithm, where ϵ depends on the ratio of the minimum capacity to maximum bandwidth of a single VC. Both [3] and [9] assume at least logarithmic ratio between maximum VC bandwidth and minimum link capacity. Similar results without this assumption were developed for special network topologies (see e.g. [10]). An optimal congestion-competitive online algorithm for multicast routing was presented in [1].

The techniques in the above mentioned papers can be used to solve several restricted multicast problems. In particular, [3] shows that if the participants in a single multicast group arrive together("batch arrivals"), and the accept/reject decision is for the whole multicast group, it is possible to achieve $O(\log n)$ competitive ratio. The case where we keep the restriction of batch arrivals, but allow rejection of some of the group members and acceptance of others is considered in [6].

Recently, Awerbuch and Singh has shown how to combine the "winner-picking" technique [2] with the techniques in [3] to achieve a polylog competitive ratio for the case where members of each multicast group arrive sequentially, i.e. the size and membership of the group is unknown upon its creation. Their algorithm can deal only with the *non-interleaved* case, i.e. when all the members of a particular multicast group arrive before a new group can be created.

It is not clear how to apply the algorithm and the analysis of [4] to the more general case, where the arrivals of requests belonging to different multicast groups are *interleaved*. The main problem is that their algorithm strongly depends on the fact that, at every instance, the algorithm is dealing with the construction of only a single multicast tree and all accept/reject decisions with respect to all existing multicast groups are already known.

As in [3], the algorithm in [4] uses edge costs the are exponential in the current link load. One of our contributions is a new definition of edge-costs that are independent of the specific accept/reject decisions made with respect to each multicast group. This decoupling between multicast groups is what allows us to apply the techniques in [3] and [2] to achieve a polylog competitive ratio for the general multicast problem.

A natural question to ask is if it is possible to make the competitive ratio independent of $\mathcal{M}$, the number of multicast group. We address this issue by showing a lower bound of $\Omega(\log \mathcal{M} \log n)$ when $\mathcal{M}$ is much larger than the link capacities.

This is the first bound for this problem that is stronger than $\Omega(\log n)$.

The algorithm presented in this paper works against a *semi-oblivious* adversary, i.e. the adversary is allowed to look at the tree used by the online algorithm to service a multicast group only after all the requests for that group have been processed[1]. We show that no algorithm can do well against an adaptive online adversary.

As is customary for online algorithms, previous papers on multicast ignored the issue of computational complexity. In particular, the algorithm in [4] assumes an NP-hard computation at each routing decision. We show that it is possible to use the special properties of the Prize-Collecting Steiner tree algorithm in [8] to implement each step of our algorithm in polynomial time.

In Section 2 we introduce the model and the terminology. Section 3 describes the algorithm, and Section 5 presents the proof of the competitive ratio. Lower bounds are presented in Section 8. Appendix A explains how to implement each decision step of our online algorithm in polynomial time.

2 Model and Definitions

A request to join a multicast group specifies the group, the node that wants to join, and the amount of the requested bandwidth. The multicast routing and admission algorithm can either reject this "join" request or accept it and allocate the requested bandwidth along some path from the new node to the current tree associated with the requested multicast group. The algorithm is not allowed to allocate above link capacity.

We model the network as a capacitated graph with n nodes and m edges. For simplicity, we will assume that all edges have capacity u and all requests are for unit bandwidth. We also assume that the number of multicast groups $\mathcal{M}$ is known in advance. The issue of removing some of the assumptions is deferred to Section 7.

We also assume that $u \geq \log \mu$, where μ is a parameter that is polynomial in $n, \mathcal{M}$ defined later.[2] We assume that multicast groups, once established, never leave. The case where each multicast group has a "holding time" will be addressed in the full version of this paper.

[1] A semi-oblivious adversary is at least as powerful as an oblivious adversary.

[2] This requirement corresponds to a similar requirement in [3].

3 The Algorithm

The online algorithm can be viewed as consisting of $L = \log n + \log \mathcal{M}$ "virtual" algorithms for each one of the $\mathcal{M}$ multicast groups. We call these algorithms virtual because the routing and accept/reject decisions of these algorithms are not implemented. Instead, they only modify internal data structures and, in particular the cost associated with each edge. The description of the cost computation is deferred to Section 4. For now, it is sufficient to assume that each edge has an associated cost that is deterministic, depends only on the input sequence of requests, and is monotonically non-decreasing in time.

The jth virtual algorithm associated with the ith multicast – $\mathrm{VA}_{i,j}$ – is shown in Figure 1. The goal of $\mathrm{VA}_{i,j}$ is to build a tree $T_{i,j}$. This tree spans some of the nodes that requested to be added to the ith multicast group and that are already spanned by trees $T_{i,k}$, for $k < i$. In other words, a request is first generated at $\mathrm{VA}_{i,1}$; if it immediately gets added to $T_{i,1}$, it is passed on to $\mathrm{VA}_{i,2}$, etc.

Each request to join the ith multicast group is considered as a potential unit of profit, and the virtual algorithms use ("consume") this profit to "pay" for their trees. $\mathrm{VA}_{i,j}$ can expand its tree $T_{i,j}$ by adding a subtree only if it can pay for this subtree. We will refer to these subtrees as "fragments". As payment, $\mathrm{VA}_{i,j}$ can use only the profit that is on the nodes of this subtree and that was not used by $\mathrm{VA}_{i,k}$ for $k < j$ (This is denoted by $\mathrm{PROFIT}_{i,j}(v)$ in Figure 1). More precisely, $\mathrm{VA}_{i,j}$ monitors the profit passed from $\mathrm{VA}_{i,j-1}$. Each time it gets π units of profit at some node v, it adds π to $\mathrm{PROFIT}_{i,j}(v)$. It then tries to find a fragment that includes v such that the ratio of the unused profit associated with nodes of this fragment plus π' is at least $d = d^T \cdot \frac{1}{6L \log n}$ times the cost of adding this fragment to $T_{i,j}$, where $d^T = 1/(mu)$ and $\pi' \leq \mathrm{PROFIT}_{i,j}(v)$. The goal of the algorithm is to minimize π'.

This subtree is added to the $T_{i,j}$, d times the cost of this tree is "consumed", and the rest of the profit (in fact, at most one unit) on the newly added nodes is bequeathed to $V_{i,j+1}$.

Observe that, since costs are increasing, the total profit used to construct $T_{i,j}$ is bounded by its final cost divided by d. Since $\mathrm{VA}_{i,j}$ builds its tree in an online fashion, there might be a larger (in terms of the spanned nodes that requested participation in ith multicast) tree that can be constructed offline using the same profit. In Lemma 5.1 we show that this "loss" is not very significant. Also, note that the only way virtual algorithms dealing with different multicast groups interact with each other is through edge costs. Another important property is that for all j, the vertices which contribute towards the profit collected by $\mathrm{VA}_{i,j}$ are a subset of the

VA$_{i,j}$

A Initialization (s_i is the root of the i-th multicast):

1 $T_{i,j} \leftarrow \{s_i\}$.

2 **for** all u

$\text{PROFIT}_{i,j}(u) \leftarrow 0$.

$\text{USED-PROFIT}_{i,j}(u) \leftarrow 0$ *($\text{USED-PROFIT}_{i,j}$ is used only for the analysis, specifically in Lemma 5.1)*

B Invoked due to receiving profit π at node v from $\text{VA}_{i,j-1}$ ($\text{VA}_{i,0}$ is invoked with unit profit due to a join request at node v):

1 $\text{PROFIT}_{i,j}(v) \leftarrow \text{PROFIT}_{i,j}(v) + \pi$.

2 Contract $T_{i,j}$ to a single vertex s.

3 Find smallest $\pi' \leq \text{PROFIT}_{i,j}(v)$ such that $\exists$ tree S with the following properties:
(i) $v \in S, s \in S$ (ii) $\text{Cost}(S) \leq (\sum_{u \in S, u \neq v} \text{PROFIT}_{i,j}(u) + \pi')/d$.

4 **if** such π' found

4.1 Let S be the tree which satisfies the conditions in Step 3.

4.2 **for** all $u \in S, u \neq v$

$\text{USED-PROFIT}_{i,j}(u) \leftarrow \text{PROFIT}_{i,j}(u)$;

$\text{PROFIT}_{i,j}(u) \leftarrow 0$.

4.3 $\text{USED-PROFIT}_{i,j}(v) \leftarrow \pi'$;
$\pi \leftarrow \text{PROFIT}_{i,j}(v) - \pi'$;
$\text{PROFIT}_{i,j}(v) \leftarrow 0$.

4.4 Uncontract $T_{i,j}$;
$T_{i,j} \leftarrow T_{i,j} \cup S$.

4.5 Update the cost of each edge $e \in S$.

4.6 Pass profit π at node v to $\text{VA}_{i,j+1}$.

Figure 1: The j-th *Virtual* phase of the i-th *Real* algorithm. Recall that d is the density value defined at the beginning of Section 3.

Real(i)

1 `Choose` $\eta_i \in [1 \ldots L]$ `such that Prob(`$\eta_i = j) = \beta \cdot 2^j$`.`

2 `Follow the computation of VA`$_{i,\eta_i}$ `and whenever an edge gets added to` T_{i,η_i}`, add that edge to the` *`real`* `tree being used to service the` i`-th multicast group.`

Figure 2: The *Real* algorithm for multicast group i.

vertices that contribute towards the profit collected by $\text{VA}_{i,j-1}$.

The "real" algorithm is shown in Figure 2. For each multicast group, it randomly chooses one of the virtual algorithms and implements the construction of the tree built by this virtual algorithm. We set the probability of choosing $\text{VA}_{i,j}$ to $p_j = \beta \cdot 2^j$, where β is chosen such that $\sum_{j=1}^{L} p_j = 1$.

Observe that if the i-th real algorithm has chosen $\text{VA}_{i,j}$ for a specific multicast, it does not get the profit for the requests that were taken into account when $\text{VA}_{i,j}$ was constructing its tree. Instead, it will get the profit for the later requests. In particular, it will get the profit that was inherited by $\text{VA}_{i,j+1}$.

In Step 3 of the virtual algorithm (see Figure 1) we need to solve the maximal dense subtree problem, which is NP-hard. In Appendix A we show how a good-enough approximation can be obtained in polynomial time.

4 Edge Costs

In this section we define the cost metric and the way it is updated as a result of each new request. The cost metric is updated by the virtual algorithms and hence is deterministic.

The online algorithm constructs the cost metric as it goes along. When profit propagates from $\text{VA}_{i,j-1}$ to $\text{VA}_{i,j}$, we consider this an "event". An event might cause $\text{VA}_{i,j}$ to consume some profit and update its tree $T_{i,j}$. Let $c_e(k)$ denote the cost of edge e after the k-th event. When the kth event occurs, the virtual algorithms use costs $c_e(k-1)$ for making their decision. These decisions are then used to compute $c_e(k)$ in a deterministic fashion.

Let $\vec{\eta} = (\eta_1, \ldots, \eta_{\mathcal{M}})$ represent the indices of the virtual algorithms chosen for the $\mathcal{M}$ multicasts. Also, let $p_{\vec{\eta}}$ represent the probability of making this sequence of choices. Define the load on an edge as $1/u$ times the number of trees it was used in by the real algorithm, and let $\lambda_e^{(\vec{\eta})}(k)$ represent the load on edge e after the first k events have occurred, where $\vec{\eta}$ represents the choices made by the *Real* algorithms. Since the random choices of the *Real* algorithms for different multicasts are independent, $p_{\vec{\eta}} = \prod_{i=1}^{\mathcal{M}} p_{\eta_i}$.

Let $c_e(0) = u$ for each edge e. Suppose costs $c(0)$ to $c(k-1)$ were already computed. Then $c_e(k)$ is computed as follows.

$$c_e(k) = u \sum_{\vec{\eta}} p_{\vec{\eta}} \mu^{\lambda_e^{(\vec{\eta})}(k)}$$

The value of μ is set to $4m^6 \log^2 \mathcal{M}$. The reason for this value will become clear in Section 6. Observe that, given $\vec{\eta}$, the expression $\lambda_e^{(\vec{\eta})}(k)$ is deterministic, and hence the costs $c_e(k)$ are deterministic as well.

Define $X_e^{(i,j)}(k)$ as indicator variables, with $X_e^{i,j}(k)$ being 1 if edge e is used by $\mathrm{VA}_{i,j}$ during the first k events and 0 otherwise. Notice that $X_e^{(i,j)}(k)$ are deterministic quantities. Now, $\lambda_e^{(\vec{\eta})}(k) = (1/u) \cdot \sum_{i=1}^{\mathcal{M}} X_e^{(i,\eta_i)}(k)$. We can use this to rewrite the cost $c_e(k)$:

$$c_e(k) = u \sum_{\vec{\eta}} \prod_{i=1}^{\mathcal{M}} \left(p_{\eta_i} \cdot \mu^{X_e^{(i,\eta_i)}(k)/u} \right)$$

Interchanging the order of the summation and the product, we get

$$c_e(k) = u \prod_{i=1}^{\mathcal{M}} \sum_{j=1}^{L} p_j \mu^{X_e^{(i,j)}(k)/u} \tag{1}$$

The above representation gives an easy way to compute $c_e(k)$ efficiently. Since only one of the sums changes during any event, the online algorithm can recompute that sum and obtain the new costs.

The following claim follows from the way we construct the cost metric.

Claim 4.1 *The cost $c_e(k)$ is the expectation of the quantity $u\mu^{\lambda_e(k)}$ where $\lambda_e(k)$ is a random variable representing the load on edge e after k events.*

5 Proof of competitiveness

In order to prove the competitive ratio, we will divide the multicast groups into "profitable" and "unprofitable", based on the cost of the *optimum* trees for these groups with respect to the cost metric constructed by our algorithm. Here, by optimum trees we mean the trees constructed by the optimum offline algorithm.

Consider the ith multicast group, and let the number of requests satisfied by the optimal offline algorithm be $r^*(i)$. Similarly, let $r(i)$ be the profit obtained by the online algorithm. Let $w^*(i)$ be the cost (in the final cost metric) of the tree T_i^* used by the optimum algorithm to service multicast group i. We call a multicast group *profitable* if the optimal's tree for this multicast group has a high profit to cost ratio in the *final* cost metric:

Definition 1 *The i-th multicast group is* profitable *if $\frac{r^*(i)}{w^*(i)} \geq d^T$, where $d^T = \frac{1}{mu}$.*

We use the quantities R^* and R to represent $\sum_{i=1}^{\mathcal{M}} r^*(i)$ and $\sum_{i=1}^{\mathcal{M}} r(i)$, respectively. Let P and U represent the set of profitable and unprofitable multicast groups, respectively. Also, we define $R_P^* = \sum_{i \in P} r^*(i)$ and $R_U^* = \sum_{i \in U} r^*(i) = R^* - R_P^*$.

We first show (Lemma 5.2) that the online algorithm obtains almost as much profit from profitable groups as the optimal solution. Then we show that the total profit obtained by the online algorithm can only be poly-logarithmically smaller than optimal's profit from unprofitable groups. To prove the latter claim, we take an indirect route. We use capacity constraints to argue that the quantity mR_U^* is bounded by the sum of the final costs of all edges (Lemma 5.3). Finally, we bound the final costs in terms of the expected profit obtained by the online algorithm (Lemma 5.7).

Consider the quantities $\textsc{Profit}_{i,j}(v)$ and $\textsc{Used-Profit}_{i,j}(v)$ at the end ie. after all requests have been received. Let $\mathrm{P}_{i,j}(v) = \textsc{Profit}_{i,j}(v) + \textsc{Used-Profit}_{i,j}(v)$. The quantity $\mathrm{P}_{i,j}(v)$ denotes the profit consumed by $\mathrm{VA}_{i,j}$ at node v. For any set X of vertices, $\mathrm{P}_{i,j}(X) = \sum_{v \in X} \mathrm{P}_{i,j}(v)$. The definitions of $\textsc{Profit}_{i,j}$ and $\textsc{Used-Profit}_{i,j}$ are similarly extended.

Lemma 5.1 $\mathrm{P}_{i,j}(T_i^*) \leq 3w^*(i)d \log n$.
Proof: O ur proof of this claim is different from the one presented in [4].

We first bound the quantity $\textsc{Profit}_{i,j}(T_i^*)$. This contribution comes from nodes in T_i^* which do not belong to $T_{i,j}$. The profit consumed on these nodes by $\mathrm{VA}_{i,j}$ must be at most $w^*(i)d$, else these nodes would have formed a fragment on their own and been added to $T_{i,j}$.

Now we bound $\textsc{Used-Profit}_{i,j}(T_i^*)$. This contribution comes from nodes that belong to $T_{i,j}$. Recall that $VA_{i,j}$ acquires $T_{i,j}$ in tree fragments. Consider an Eulerian tour D of T^*. Let a *segment* of tour D be a maximal contiguous piece of

D such that all edges of the segment belong to the same fragment of $T_{i,j}$. Initially, all segments are marked active. If two consecutive active segments on this tour belong to the same fragment, they are merged together along with the portion of the tour between between them to form a single segment. Let $t(s)$ denote the event at which the edges of segment s were added to $T_{i,j}$.

Furthermore, we define a $pred$ and $succ$ relation on active segments such that $pred(s, D)$ is the predecessor of s in tour D and $succ(s, D)$ is the successor of s in D.

Let $D_0 = D$. For $h \geq 1$, let $\mathcal{H}_h = \{s$ is an active segment of $D_{h-1}, t(s) < t(pred(s, D_{h-1})), t(s) < t(succ(s, D_{h-1}))\}$. Let $\mathcal{L}_h$ denote the remaining segments of D_{h-1}, and let D_h denote the tour D_{h-1} with each segment in $\mathcal{L}_h$ marked inactive. The segments in $\mathcal{H}_h$ remain active in D_h. As mentioned above, consecutive active segments are merged if they belong to the same fragment.

Note that for all h:

$$|\mathcal{L}_h| > |\mathcal{H}_h|.$$

This implies that there at most $\log n$ non empty sets $\mathcal{L}_h$. Let $s \in \mathcal{L}_h$ for some h. Also, let s' be the successor or predecessor segment of s in D_{h-1} with $t(s') < t(s)$ and let p consists of the part of D between s and s'.

Assume $\text{USED-PROFIT}_{i,j}(s) \geq d(w^*(s) + w^*(p))$. Let $v \in s$ be the node with the last request in multicast i among all nodes in s. When the request at v arrived, the sum $\text{PROFIT}_{i,j}(s) = \sum_{u \in s} \text{PROFIT}_{i,j}(u)$ is at least $d(w^*(s) + w^*(p))$, because $\text{PROFIT}_{i,j}(s)$ is the source of $\text{USED-PROFIT}_{i,j}(s)$. Thus, at that time we could have used at most $d(w^*(s) + w^*(p))$ to add $s + p$ as a fragment. Since the algorithm always tries to create a fragment using the minimum amount of profit, we have:

$$\text{USED-PROFIT}_{i,j}(s) \leq d(w^*(s) + w^*(p)).$$

Considering that D visits every node twice it follows that

$$\sum_{s \in \mathcal{L}_h} \text{USED-PROFIT}_{i,j}(s) \leq 2w^*(i)d.$$

Summing over all values of h it follows that the profit consumed by $VA_{i,j}$ from all nodes which belong to $T^* \cap T_{i,j}$ is at most $2w^*(i)d \log n$. This completes the proof of this lemma. ■

It is easy to see that, for profitable multicasts, the profit obtained by our online algorithm is high:

Lemma 5.2 $R \geq R_P^*/2$.

Proof: Since there are L levels, Lemma 5.1 guarantees that the total wasted profit for multicast group i is at most $3Lw^*(i)d\log n$. Plugging in $d = d^T \cdot \frac{1}{6L\log n}$ and using the fact that i profitable implies that $r^*(i) \geq d^T w^*(i)$, we obtain a bound of $r^*(i)/2$ on the wasted profit. Therefore, $r(i) \geq r^*(i)/2$ for all profitable groups i. Summing over all the profitable groups, we get the desired result. ■

Having bounded the profit from the profitable groups, we now concentrate on the unprofitable groups. Recall that c_e is the cost of edge e at the end i.e. after all the events have taken place, and that the costs are non-decreasing in time.

Lemma 5.3 $mR^*_U \leq \sum_e c_e$

Proof: Let k^*_e be the number of multicast groups which use edge e in the optimal offline solution. Consider the tree T^*_i used by the optimal to route the i-th multicast group. If this group is unprofitable then by definition $r^*(i) \leq \frac{1}{mu}\sum_{e\in T^*_i} c_e$. We sum this over all the unprofitable multicast groups, and then reverse the order of summation.

$$\begin{aligned} R^*_U &\leq \frac{1}{mu}\sum_{i\in U}\sum_{e\in T^*_i} c_e \\ &\leq \frac{1}{mu}\sum_{e} k^*_e c_e \\ &\leq \frac{1}{m}\sum_{e} c_e \end{aligned}$$

The last inequality follows from the fact that the optimal offline is not allowed to exceed capacities, implying $k^*_e \leq u$. ■

Let $w_j(i)$ represent the cost incurred by $\mathrm{VA}_{i,j}$ in constructing the tree $T_{i,j}$. In other words, each tree fragment of $T_{i,j}$ contributes to $w_j(i)$ its cost associated with the event of adding this fragment. We use $w(i)$ to denote $w_{\eta(i)}(i)$, where η represents the chose of the real algorithm. Let $r_j(i)$ represent the profit consumed in constructing this $T_{i,j}$. The following lemma implies that if the expected profit is small, then the expected cost of the constructed trees is small as well.

Lemma 5.4 $\mathbf{E}(r(i)) \geq (d/2)\mathbf{E}(w(i)) - \frac{1}{\mathcal{M}}$, *where* $w(i)$ *is the cost paid by the* Real *algorithm for multicast group* i.

Proof: If the real algorithm chooses to follow $\mathrm{VA}_{i,j}$, i.e. $\eta(i) = j$, then it will get at least the profit used by $\mathrm{VA}_{i,j+1}$. Therefore:

$$\mathbf{E}(r(i)) \geq \sum_{j=1}^{L-1} p_j r_{j+1}(i).$$

By definition, $p_j = p_{j+1}/2$, and hence

$$\mathbf{E}(r(i)) \geq \sum_{j=2}^{L} p_j r_j(i)/2.$$

By construction:

$$\mathbf{E}(w(i)) = \sum_{j=1}^{L} p_j w_j(i) \leq (1/d) \sum_{j=1}^{L} p_j r_j(i).$$

Thus, we have

$$d \cdot \mathbf{E}(w(i)) \leq 2\mathbf{E}(r(i)) + p_1 r_1(i).$$

Now notice that $r_1(i)$ can be at most n, since each request brings in one unit of profit, and there can be at most n requests for a single multicast group. Also, $p_1 \sum_{j=1}^{L} 2^{j-1} = 1$, which implies that $p_1/2 < 2^{-L}$. Substituting $L = \log n + \log \mathcal{M}$, we obtain $\mathbf{E}(r(i)) \geq (d/2)\mathbf{E}(w(i)) - \frac{1}{\mathcal{M}}$.

We remark that in this proof the fact that the VAs are deterministic is quite crucial; otherwise, the profits $r_j(i)$ would be conditioned on the random choices made by the real algorithms and the above argument would break down completely.

■

Now we prove that if the expected cost of the constructed trees is small, then the total cost of all the edges is small as well. But first, we need to prove the following technical lemma. Roughly speaking, this lemma implies that if an event caused an edge to be used by one of the trees, the increase in the cost of this edge is proportional to its current cost.

Consider an event k that caused $\mathrm{VA}_{i,j}$ to augment its tree, and let E_k represent the set of edges of the newly added subtree.

Lemma 5.5 *For all* $e \in E_k$, $c_e(k) - c_e(k-1) \leq \frac{\log \mu}{u} p_j c_e(k-1)$. *For the edges* $e \notin E_k$, $c_e(k) = c_e(k-1)$.

Proof: The second part of the lemma is obvious. We concentrate on edges $e \in E_k$. By definition of the indicator variables, $X_e^{(i,j)}(k-1) = 0$ and $X_e^{(i,j)}(k) = 1$. Using Equation 1, we have:

$$\begin{aligned} c_e(k) - c_e(k-1) &= p_j(\mu^{X_e^{(i,j)}(k)/u} - 1)u \prod_{i' \neq i} \sum_{j'} p_{j'} \mu^{X_e^{(i',j')}(k)/u} \\ &= p_j(\mu^{1/u} - 1) \frac{c_e(k-1)}{\sum_{j'} p_{j'} \mu^{X_e^{(i,j')}(k-1)/u}} \\ &\leq p_j(\mu^{1/u} - 1) c_e(k-1). \end{aligned}$$

The last inequality above follows from the fact that $\sum_{j'} p_{j'} \mu^{X_e^{(i,j')}(k-1)/u} \geq \sum_{j'} p_{j'} = 1$. For all x between 0 and 1, $2^x - 1 \leq x$. Therefore, $\mu^{1/u} - 1 = 2^{(\log \mu)/u} - 1 \leq (\log \mu)/u$, which completes the proof of the lemma. ■

Let $W = \sum_i w_i$ represent the total cost of the trees constructed by the online algorithm. The following lemma relates the cost incurred by the algorithms and the final cost of the edges.

Lemma 5.6 $\frac{\log \mu}{u} \mathbf{E}(W) \geq \sum_e (c_e - u)$.
Proof: Let $\Delta_e(k) = c_e(k) - c_e(k-1)$ represent the increase in cost on edge e during the kth event. Clearly, $c_e = c_e(0) + \sum_k \Delta_e(k)$ where the summation is over all events and $c_e(0) = u$ for all edges e. Now, let VA_{i_k, j_k} be the virtual algorithm that updates its tree during event k. Lemma 5.5 implies that

$$\sum_e (c_e - u) \leq (\log \mu/u) \sum_i \sum_j p_j \sum_{k: i_k = i, j_k = j} \sum_{e \in E_k} c_e(k-1).$$

Using definition of $w_j(i)$, we can rewrite this expression as follows:

$$\sum_e (c_e - u) \leq (\log \mu/u) \sum_i \sum_j p_j w_j(i) = (\log \mu/u) \sum_i \mathbf{E}(w(i)).$$

Using linearity of expectations, $\sum_i \mathbf{E}(w(i)) = \mathbf{E}(\sum_i w_i))$, which completes the proof. ■

We are now ready to show that if the obtained profit is small, then the total cost of all the edges is small as well.

Lemma 5.7 $(5\frac{d^T}{d} \log \mu) m \mathbf{E}(R) \geq \sum_e c_e$
Proof: Summing up Lemma 5.4 over all multicast groups, we have:

$$\frac{2}{d} \left(\mathbf{E}(R) + \sum_{i=1}^{\mathcal{M}} \frac{1}{\mathcal{M}} \right) \geq \mathbf{E}(W).$$

As we will show below, $\mathbf{E}(R) \geq 1$. Therefore, the above inequality can be rewritten as $\frac{4}{d} \mathbf{E}(R) \geq \mathbf{E}(W)$. Using Lemma 5.6, and the fact that $m d_T u = 1$, we obtain

$$\begin{aligned} \sum_e c_e &\leq \frac{\log \mu}{u} \cdot \frac{4}{d} \mathbf{E}(R) + mu \\ &= m \log \mu \left(\frac{4 d_T}{d} \mathbf{E}(R) + \frac{u}{\log \mu} \right). \end{aligned}$$

To complete the proof, it remains to show that the first $u/\log\mu$ requests are always accepted, i.e. $\mathbf{E}(R) \geq u/\log\mu$. Suppose the first $k < u/\log\mu$ requests have been accepted. As a result, the load on each edge is no more than k, and the cost of servicing the next request can be at most $mu\mu^{k/u} < mu\mu^{\frac{1}{\log\mu}} = 2mu$. By construction, the profit needed to pay for this cost is at most

$$2mud = 2mu\frac{1}{mu}\frac{1}{6L\log n} = \frac{1}{3L\log n}$$

Thus, the unit of profit brought by this request is enough to pay for extending the trees of all VA algorithms dealing with the corresponding multicast group. Thus, this request is going to be accepted by the real algorithm as well. In other words, if there are less than $u/\log\mu$ requests generated by the adversary then the *Real* algorithm accepts them all and has a competitive ratio of 1. Else, R (and therefore $\mathbf{E}(R)$) is greater than $u/\log\mu$, which completes the proof of the claim. ■

Combining Lemma 5.3 and Lemma 5.7 with Lemma 5.2, we obtain the following result:

Theorem 1 $R^*/\mathbf{E}(R) = O(\log n \log\mu(\log n + \log\mathcal{M}))$

6 Capacity Constraints

In the previous section we showed that the algorithm accepts a significant fraction of the requests accepted by the optimum offline algorithm. It remains to show that our online algorithm does not overflow the available capacities. To that end, we set $\mu = 4m^6\log^2\mathcal{M}$. Note that, by Theorem 1, this implies that we get an $O(\log n(\log n + \log\log\mathcal{M})(\log n + \log\mathcal{M}))$-competitive algorithm. For the special case where each node is allowed to serve at most one multicast group, we clearly have an $O(\log^3 n)$-competitive algorithm.

We now show that the above value of μ is sufficient to ensure that the capacity constraints are never violated with high probability.

Lemma 6.1 *For any edge e, the cost c_e does not exceed $u\mu^{1/2}$.*

Proof: Suppose $c_e(k) > u\mu^{1/2-1/u}$ for some k. Since $\mu = 4m^6\log^2\mathcal{M}$ and $u \geq \log\mu$, we get $c_e > um^3\log\mathcal{M}$. Since maximum profit of a single tree fragment is n, this cost is above maximum profit divided by d. Thus, this edge will never be used again by any VA. The claim follows from the fact that during any one event, $c_e(k)$ can increase only by a factor of $\mu^{1/u}$. ■

Lemma 6.2 *With probability at least $1 - 1/m^2$, no edge violates its capacity constraint.*

Proof: Claim 4.1 states that c_e is equal to the expected value of the quantity $u\mu^{\lambda_e}$, where λ_e is the final load on an edge. The event $\lambda_e \geq 1$ implies that $u\mu^{\lambda_e} \geq \mu^{1/2}\mathbf{E}(u\mu^{\lambda_e})$. Using Markov inequality, the probability of this event happening is at most $\mu^{-1/2} < 1/m^3$. Therefore, with probability at least $1 - 1/m^2$, all edges satisfy the capacity constraints. ■

If the algorithm tries to exceed capacity of an edge, we terminate it. Lemma 6.2 guarantees that this does not affect the competitive ratio given in Theorem 1.

7 Relaxing some of the assumptions

Using techniques in [3], it is easy to relax some of the assumptions made in Section 2. In particular, we can allow the bandwidth requirements of different multicast groups and the capacities of the edges to vary arbitrarily, as long as the largest bandwidth requirement is no more than $\frac{1}{\log \mu}$ times the smallest edge capacity. Also, different multicast groups can have different profits as in [3]. Further, we do not need to know $\mathcal{M}$ in advance. We can keep doubling our guess of $\mathcal{M}$, modifying μ accordingly; this results in the loss of another factor of $\log \mathcal{M}$ in the competitive ratio.

We would like to point out that this competitive ratio holds against a *semi-oblivious* adversary – the adversary is allowed to look at the multicast tree generated by the online algorithm but only after all the requests for that multicast group have been processed. The next obvious question to ask is whether any algorithm can work well against a more powerful adversary. We answer this question in the negative in Section 8.2

8 Lower bounds

8.1 Against an oblivious adversary

A lower bound of $\log n$ for the problem studied in this paper immediately follows from [3]. The challenge in the online multicast problem is to decide which requests to service ("winner picking") and how to route a request ("online routing"). We now show how to combine a lower bound for winner picking [2] with the lower bound for online routing [3] to achieve a lower bound of $\log \mathcal{M} \log n$ for the online multicast problem. This is the first lower bound stronger than $\log n$ for the online multicast problem.

Theorem 2 *No algorithm for selective online multicast can have a competitive ratio better than* $\Omega(\log(\mathcal{M}/u)\log n)$ *even against an oblivious adversary, and even when the requests are non-interleaved.*

Proof: The basic idea behind the winner picking lower bound for online multicast is the following: Assume $\mathcal{M}$ multicasts are created, but both the online and the offline algorithm are just allowed to pick one. A multicast consists of at least one and up to $\log \mathcal{M}$ classes, each class consisting of c requests for some parameter c. Half of the multicasts, chosen randomly from all multicasts, consist of exactly c request. One fourth of the multicasts, chosen randomly from the remaining half of the multicasts, consist of exactly $2c$ request, etc. Thus, the expected profit of online is $2c$, while the expected profit of offline is $c \log \mathcal{M}$.

The lower bound for online routing works in phases: There are $\log n + 1$ phases, with the "profit", i.e. number of requests, doubling in each phase. It can be shown that there must be a phase such that the expected profit that online has received so far is at most $2/\log n$ of the profit that is available in the current phase. In this phase, offline services all the request, i.e., takes all the profit, and the sequence of requests terminates.

We show next how to combine these two bounds. To simplify the presentation we assume that all demands and all edge capacities are 1, but it is permissible to satisfy a fractional demand and obtain a fractional profit (the profit for a multicast group is the product of the satisfied demand and the number of satisfied requests). We explain later how this result carries over to our model.

We restrict ourselves to values of $\mathcal{M}$ such that $\sqrt{n} > \log \mathcal{M}$. Consider the graph G on $n+2$ vertices (see Figure 3) which is defined as follows. The vertex set is $\{r, x, v_1, \ldots, v_n\}$. There is an edge from r to x, and there is an edge from x to each of $v_1 \ldots v_n$. For convenience, define $M = \mathcal{M}/\log n$ and $N = n/\log M$. Notice that the restriction we have placed on $\mathcal{M}$ implies that $N > \sqrt{n}$.

The adversary operates in at most $\log N$ phases: we describe the i-th phase, $1 \le i \le \log N$. In phase i the adversary divides the vertices $v_1 \ldots v_n$ into classes of size 2^{i-1}. Notice that there must be at least $\log M$ classes. The adversary then generates M multicasts, each with r as the root. The requests for these multicasts will be non-interleaved. For each multicast, the adversary generates a request at each of the nodes in the first class. Then the adversary flips a coin. If the coin toss is a Head (ie. with probability half) the adversary moves on to the next multicast. Else, it generates a request at each node in the next class, flips another coin, and

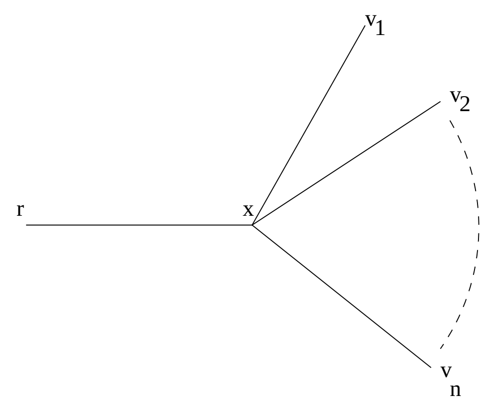

Figure 3: The lower bound graph for Theorems 2 and 3

repeats the same process again. If requests have been generated at $\log M$ classes for the same multicast, the adversary moves on to the next multicast. At the end of all M multicasts for this phase, the adversary moves on to the next phase. Notice that setting the class size to 2^{i-1} is equivalent to doubling available profit by 2 for each phase.

Let $c(i)$ be the capacity on the edge (r, x) used by the online algorithm during phase i. Also, let $p(i)$ be the profit obtained by online during the i-th phase. Let $p^*(i)$ and $c^*(i)$ be the corresponding quantities for the solution generated by the oblivious adversary. Notice that $\sum_i c(i)$ can be at most 1. Define $S(k) = \frac{1}{2^k}\sum_{1\leq k}\mathbf{E}(p(i))$. The total expected profit obtained by the algorithm in the first k phases is $2^k S(k)$.

We give brief proofs for the following two claims.

Claim 8.1 $\mathbf{E}(p(i)) < 2^i \mathbf{E}(c(i))$.

Proof: Suppose the online algorithm decides to satisfy a fractional demand of x for a specific multicast in the i-th phase. The cost incurred is x. Suppose that this commitment is made by the algorithm after the j-th request for this multicast group comes in. Then the expected profit from this multicast group is $2^{i-1} \cdot x\sum_{j\leq j'\leq M} 2^{j'-j} < 2^i x$. Now we sum this up over all the multicast groups in phase i to get the desired result. ■

Claim 8.2 *During any phase* i*, the adversary can ensure that*

$$\mathbf{E}(p^*(i)) \geq \frac{\log M}{4} 2^i c^*(i).$$

Proof: During phase i, the adversary can pick the multicast with the maximum number of classes of requests. Let $P^{<}(i)$ denote the probability of this number being less than i. Now, $P^{<}(i) = (1/2 + 1/4 + \ldots 2^{1-i})^M = (1 - 2^{1-i})^M$, for $i < \log M$. Clearly, $P^{<}(\frac{2}{3}\log M) < 1/3$.[3] This tells us that the expected number of classes is greater than $\frac{\log M}{2}$. To complete the proof of this claim we observe that each class in the i-th phase has 2^{i-1} requests. ■

We now prove that there exists a phase k such that the total expected profit obtained by the online algorithm during the first k phases is no more than $2^{k+1}/\log N$. Suppose this is not true. Then, $S(i) > 2/\log N$ for all i. In particular,

$$\sum_{1 \leq i \leq \log N} S(i) > 2.$$

But

$$\sum_i S(i) = \sum_i \mathbf{E}(p(i)) \sum_{i \leq j \leq \log N} \frac{1}{2^i} \leq 2 \sum_i \mathbf{E}(p(i))/2^i.$$

Using Claim 8.1, we have

$$\sum_i \mathbf{E}(c(i)) > 1.$$

But this is a contradiction, as the online is not allowed to overflow capacities. This proves the existence of a phase k with $S(k) \leq 2/\log N$.

The oblivious adversary cannot see the coin tosses of the online algorithm but it can compute in advance the quantities $S(i)$. Having found the value k guaranteed by the above argument, the adversary stops after phase k and does not generate any more multicast requests. The adversary also generates a 'good' solution as follows: It does not satisfy any demands in the first $k-1$ phases, and in the last phase, it uses up the entire edge (r, x). Now from Claim 8.2, $\mathbf{E}(p^*(k)) \geq 2^{k-2}\log M$. The total expected profit obtained by the online algorithm is $2^k S(k) \leq 2^{k+1}/\log N$. This gives a lower bound of $\Omega(\log M \log N)$ on the competitive ratio of any online algorithm. Since $N \geq \sqrt{n}$ and $M = \mathcal{M}/\log n$, this is also a $\Omega(\log \mathcal{M} \log n)$ lower bound.

In the above analysis, we assumed that $\sqrt{n} > \log \mathcal{M}$. This is not a very restrictive assumption, because for $\log \mathcal{M} > \sqrt{n}$, our proof shows that the competitive ratio is already as bad as $\Omega(\sqrt{n})$.

Now we adapt this lower bound proof to our model. Assume that the capacity is u. Let $\mathcal{M}$ be the number of multicasts, and let $\mathcal{M}' = \mathcal{M}/u$. The adversary

[3] This is a very loose statement. $P^{<}(\frac{2}{3}\log M)$ is much,much smaller, but we do not need a stronger bound.

proceeds as before, except that each phase gets repeated u times. Also, the online algorithm is restricted to satisfy the entire demand of 1 unit or none at all. The same calculation as done above gives a lower bound of $\Omega(\log \mathcal{M}' \log n) = \Omega(\log(\mathcal{M}/u) \log n)$. ■

8.2 Against an adaptive-online adversary

Recall that an adaptive-online adversary is one which can adapt the input sequence depending on the response of the online algorithm; however the adversary must also generate a solution as it goes along.

Theorem 3 *No randomized algorithm for selective online multicast can have better than* $\Omega(\min(n, \frac{\mathcal{M}}{u}))$ *competitive ratio against an adaptive-online adversary. The lower bound holds even when the requests are non-interleaved.*

Proof: Consider the same graph as in the previous section (see Figure 3). Each edge has a capacity u, all requests have demand 1. The value of $\mathcal{M}$ is fixed at $2nu$. The adversary works in $2u$ phases.

During the i-th phase, the adversary chooses a number m_i uniformly at random from the set $\{1, \ldots, n\}$. The adversary then does the following for at most $m_i - 1$ sub-phases. It generates a new SET-UP request at r. It then generates, in sequence, requests at nodes $v_1, \ldots, v_n$ for the newly set up multicast group. As soon as the online algorithm accepts any of these requests, the adversary aborts this phase completely and moves to the next phase. If the online algorithm does not accept any of these n requests, the adversary moves to the next sub-phase. The adversary does not accept any request during the first $m_i - 1$ sub-phases. If all $m_i - 1$ sub-phases end without the online algorithm having accepted even one request, then the adversary generates yet another SET-UP request at node r. It then generates requests in sequence at nodes $v_1, \ldots, v_n$ and accepts them all without bothering about the online algorithm (if the edge $< r, x >$ is already full in the adversary's network, the adversary rejects these requests). During any phase, the online algorithm gets a profit of n if it 'guesses' correctly the value m_i, and at most 1 otherwise. Since m_i is chosen uniformly at random from $\{1, \ldots, n\}$, the expected profit for the online algorithm during any one phase is at most 2.

At the end of these $2u$ phases, the adversary is guaranteed to have accepted un requests. The expected number of requests accepted by the online algorithm can be no more than $4u$. This proves the desired result. Notice that the requests generated by the adversary are non-interleaved. If each node is allowed to serve no more than one multicast group, the above argument can be modified to obtain a $\Omega(\sqrt{n})$ lower bound for $u = O(\sqrt{n})$. ■

References

[1] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. *25th ACM Symposium on Theory of Computing*, pages 623–31, 1993.

[2] B. Awerbuch, Y. Azar, A. Fiat, and T. Leighton. Making commitments in the face of uncertainty: How to pick a winner almost every time. *28th ACM Symposium on Theory of Computing*, pages 519–530, 1996.

[3] B. Awerbuch, Y. Azar, and S. Plotkin. Throughput competitive online routing. *34th IEEE symposium on Foundations of Computer Science*, pages 32–40, 1993.

[4] B. Awerbuch and T. Singh. Online algorithms for selective multicast and maximal dense trees. *29th ACM Symposium on Theory of Computing*, 1997.

[5] N. Garg. A 3-approximation for the minimum tree spanning k vertices. *37th IEEE Symposium on Foundations of Computer Science*, pages 302–309, 1996.

[6] A. Goel and S. Plotkin. An algorithm for throughput competitive online multicasting in capacitated networks. *Unpublished Manuscript*.

[7] M. Goemans and J. Kleinberg. An improved approximation ratio for the minimum latency problem. *7th ACM-SIAM Symposium on Discrete Algorithms*, pages 152–158, 1996.

[8] M. Goemans and D. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296–317, April 1995.

[9] A. Kamath, O. Palmon, and S. Plotkin. Routing and admission control in general topology networks with poisson arrivals. *7th ACM-SIAM Symposium on Discrete Algorithms*, pages 269–278, 1996.

[10] J. Kleinberg and E. Tardos. Disjoint paths in densely embedded graphs. *36th IEEE symposium on Foundations of Computer Science*, pages 52–61, 1995.

[11] D. Sleator and R. Tarjan. Amortized efficiency of list update and paging rules. *Comm. of the ACM*, 28(2):202–208, 1985.

A Making our algorithm run in polynomial time

There are two issues we need to address to make our algorithm run in polynomial time. First, in step 3 of $\text{VA}_{i,j}$ (Figure 1), we need to compute the minimum profit needed out of the new request to create an appropriate tree fragment. Second, we need to provide a polynomial time approximation algorithm for finding the fragment in step 3 of $\text{VA}_{i,j}$.

The first problem can be solved by doing a binary search. Let π be the available profit on the node where the new request has been generated. We run a maximal dense tree algorithm with all π units of profit available. If a non-empty tree is returned, we try with half the profit, and so on. The key observation here is that we can afford to incur an additive error of $1/(8nL)$ in finding the minimum profit π'. Therefore, the binary search needs to be done only up to a depth $\log 8\pi nL$. The competitive ratio remains unaffected by this modification.

The second problem is resolved using the prize collecting Steiner tree algorithm of Goemans and Williamson [8] which has some very nice properties [7, 5].The prize collecting Steiner tree problem is the following: given a cost metric on the edges, and a penalty function defined on the vertices, find a tree such that the sum of the cost of the edges in the tree and the penalty of the vertices not in the tree is minimum. We already have a cost metric defined on edges. For a vertex v with available profit π_v, the penalty of missing v is set to π_v/d. Now the approximation algorithm [8] for the prize collecting Steiner tree problem is invoked, and the tree fragment returned by this algorithm is used. If d is set to half its current value, we obtain the same bound on the competitive ratio (except the loss of a factor of two). The proof goes along the lines of [6], where the same technique is used in the context of semi-selective online multicast.