Distributed Admission Control, Scheduling, and Routing with Stale Information

Ashish Goel [*]
University of Southern California

Adam Meyerson [†]
Stanford University

Serge Plotkin [‡]
Stanford University

October 6, 2000

Abstract

We study the problem of distributed online admission control and routing of permanent virtual circuits in a capacitated network. We assume that we have k distinct decision makers, each of which is responsible for gathering its own information about the state of the network. Through simulation, we demonstrate that an exponential based routing scheme will perform well in a distributed model provided granularity is sufficiently high. In order to ground these results theoretically, we prove that exponential-based schemes attain best-possible competitive ratios (same as for the centralized case) provided each edge can accommodate at least $\Omega(k \log n)$ requests. A matching lower-bound shows that no deterministic algorithm can attain best-possible competitive ratios without requiring the same level of granularity. In the randomized case, we present a modified exponential-based approach which obtains best-possible competitive ratios provided the granularity is at least $\Omega(k + \log n)$.

Our results may be extended to the case where different requests have different profits, and where requests are allowed to be temporary. They also apply to admission control and scheduling for unrelated machines.

[*]Department of Computer Science, University of Southern California. Email: agoel@cs.usc.edu.
[†]Department of Computer Science, Stanford University CA 94305. Email: awm@cs.stanford.edu.
[‡]Department of Computer Science, Stanford University CA 94305. Email: plotkin@cs.stanford.edu.

1 Introduction

The problem of routing and admission control in a capacitated network has been well studied when there is a centralized decision maker [2, 10, 8, 5, 6, 9]. In large networks, it is unreasonable to assume the existence of a single centralized routing agent with full instantaneous knowledge of the state of the network. In this paper we study the case where the routing function is not centralized but distributed among k different routing agents. Each call setup request goes to one routing agent, which must then route it without recourse to information local to other routing agents and without first exploring the network. We allow routers to gather information by "piggybacking;" each time a request is sent along a path, the router for that request receives information about the current congestion of the links along this path.

Our simulations demonstrate that an exponential-based routing scheme [5] performs well even in the distributed case. We observe that distributed performance is very close to centralized performance when the network links can handle a large number of requests (high network granularity). Distributed performance degrades as the requests are made larger relative to the link capacities (decreased granularity). Intuitively, the use of stale information about the network causes certain mistakes to be made by the routing agents. The number of such mistakes which can occur on any given edge grows linearly with the number of routers. If the link capacity is high relative to the number of routing agents, the mistakes will not have a significant effect on overall performance. On the other hand, as we reduce the link capacity, the mistakes will start to impact performance to a greater degree.

We formalize this notion using the concept of competitive ratio – the worst case ratio of optimum offline profit to online profit obtained (in the simple sense, profit is the number of requests routed; we could give the requests different priorities, in which case profit would be the sum over requests routed of the priority). Centralized online routing has been studied in detail [8, 10, 5], leading to matching upper and lower bounds of $O(\log \mu)$ online competitiveness. Here μ is the product of the maximum length path in the graph, the maximum request profit, and the maximum request duration (minimum profits and durations are scaled to be one). Since the use of many distributed routers with stale information cannot make the problem easier, the lower bound results continue to hold. We prove that given granularity of $\Omega(k \log \mu)$ requests per link, the distributed version of the exponential algorithm achieves $O(\log \mu)$ competitivity. This correlates with our simulation results in suggesting that, provided granularity is sufficiently high, distributed routing performs as well as centralized routing. On the other hand, we present lower bounds indicating that without granularity $\Omega(k \log \mu)$, no deterministic distributed algorithm can obtain $O(\log \mu)$ competitivity in the worst case.

A randomized algorithm can obtain $O(\log \mu)$ competitivity when the granularity is only $\Omega(k + \log \mu)$. We present a randomized algorithm which matches this bound, and provide matching lower bounds. While our randomized algorithm is not very practical (in effect, we sacrifice our $O(\log \mu)$ competitive ratio up front), it does give some indications as to the behavior of a deterministic algorithm on random traffic. The result suggests that the required granularity for effective distributed routing will not be so high as $\Omega(k \log \mu)$ in a network where the traffic is not adversarial in nature.

Similar problems arise in the context of unrelated machine scheduling, where each scheduling agent maintains its own estimate of the load on each machine; this estimate is updated each time the agent tries to schedule a job on that machine. Machine scheduling can be viewed as routing on a star where all requests specify the center as their source and a subset of the other nodes as feasible sinks. Our results (both lower and upper bounds) will therefore hold for machine scheduling as well (since all paths on a star have length at most two, we will have $\mu = 2p_{max}\tau_{max}$).

1.1 Previous Work

The dual problem of minimizing congestion in a network was studied by [2] in the centralized case. Distributed algorithms for this problem were presented by [11] following a model described by [7, 12]. Online algorithms with stale information were also studied by [1] in the context of paging and metrical task systems. Awerbuch *et al.* [3] also studied the problem of distributed resource allocation. The main difference between our model and the one in [3] is that we do not assume up-to-date knowledge of the state of the network and we explore only one path at a time.

The twin problems of multicast routing and admission control were studied by [6, 9] who gave polylogarithmic competitive algorithms for this problem. Congestion control for multicast circuits was studied by [4]. While we do not explicitly study multicast admission control in this paper, we believe that our techniques should extend to multicast routing and admission control in a straight forward fashion, resulting in a distributed version of the algorithms in [6, 9].

2 Simulations

2.1 Simulation Parameters

The simulations were performed on a 25 node network with 45 bidirectional links of capacity 150 Mbps, 5 of 300 Mbps, and 2 of 450 Mbps. Holding times for calls are exponential with a mean of 30 minutes. Each call requires 1.1 Mbps in the low granularity case and **10 times less** in the high granularity case. We plot the rejection rate for different arrival rates (in calls per second). We call this network FRAME since this is a real frame relay network topology. Results on other topologies are similar, and are omitted from this paper.

Notice that $k \log m$ for this network is 150, and the two granularities we simulate are approximately 140 and 1400.

In these simulations, we assume that each node knows the exact state of links incident on it. Further, if the information a node has about a link is more than 10 minutes old, the node disregards this information and assumes that the edge is empty.

2.2 Simulation Results

We plot the percentage of requests rejected (y-axis) against the frequency of requests (x-axis). We compare the distributed and centralized versions of both the exponential and shortest-path based algorithms. Figure 2 displays the results for a very granular network (request bandwidths are small relative to capacities) while figure 1 displays the results for a less granular network (request bandwidths are larger, frequencies decreased to keep the overall load the same). The solid lines represent centralized routing; dotted lines represent distributed routing. The exponential routing schemes are denoted by asterix.

Observe that the exponential-based scheme performs significantly better than the shortest-path approach. This correlates with previous results in [2]. For the high-granularity cases, the distributed

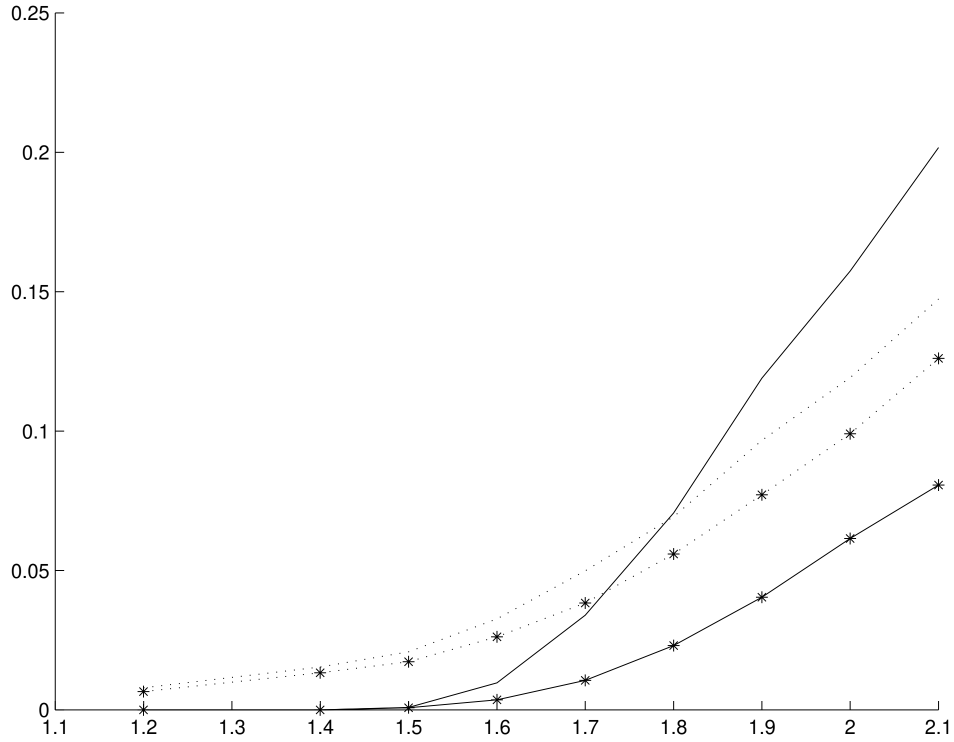

Figure 1: FRAME with Low Granularity

and centralized exponential approaches perform nearly identically. For low-granularity, distributed shortest-path and exponential have much the same performance; with frequently stale information and low granularity, both distributed routing schemes become essentially random. We note that "random" routing performs better than shortest path in extremely loaded networks, explaining the somewhat surprising result that distributed shortest-path outperforms centralized shortest-path.

3 Theoretical Model

We model the network as a graph $G = (V, E)$ with n nodes and m edges, along with capacities on the edges. We will denote the capacity on edge $e \in E$ by u_e. We are also given k routing agents (routers).

Requests for call setup will arrive in an online fashion. Each request i specifies a source node ($s_i \in V$) and a set of feasible sink nodes ($T_i \subseteq V$). Each request has integer start and finish times (τ_i^s and τ_i^f with $\tau_i^f > \tau_i^s$). Every request has a profit value $p_i \geq 1$. Finally, each request is made to a specific router r_i. We can thus represent requests with six-tuples; a request is of the form:

$$(s_i, T_i, \tau_i^s, \tau_i^f, p_i, r_i)$$

.

Allowing multiple sink nodes differs from the standard model [5, 2] but a straightforward extension of the same algorithm will handle multiple sinks in the single router case. This extension

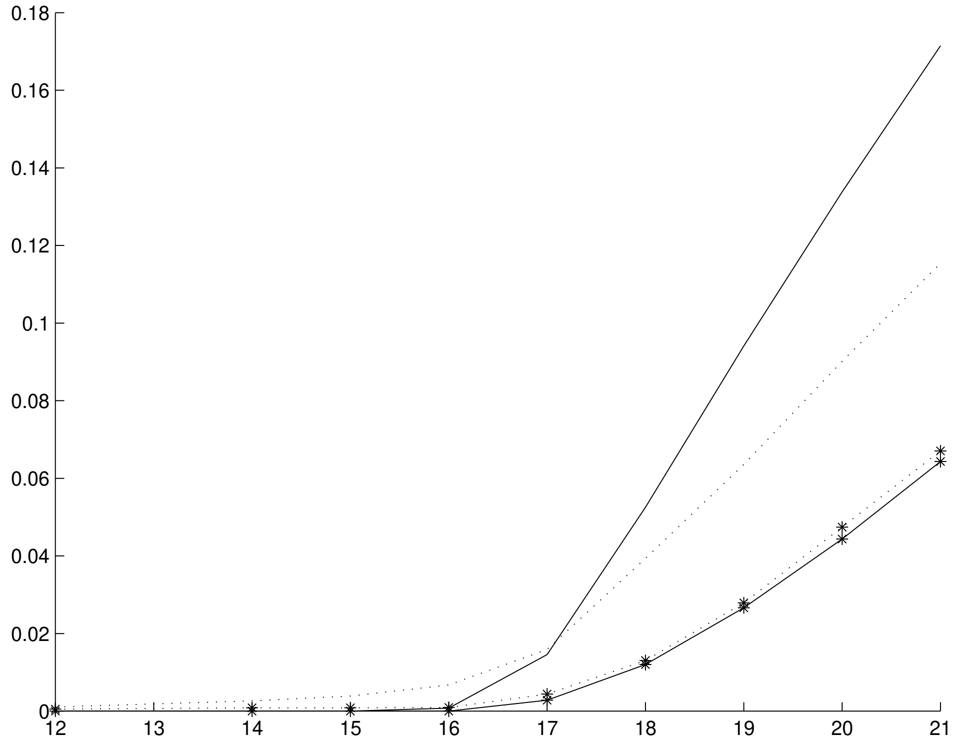

Figure 2: FRAME with High Granularity

models the case in which an individual might wish to connect to one of many nodes in order to download a file or join a multicast tree.

The routers each maintain their own image of the network state. When a request arrives to a router, the router must determine whether to route or reject that request using the request specifications and the router's network image. If the request is to be routed, the router must also determine a path from s_i to some sink $t_i \in T_i$ along which to set up the virtual circuit. Once these determinations are made, both the network state and router r_i's network image are updated; we are allowed to use examine the actual network state at this point in order to update the local image.

There are two important things to notice about this. First, the decision of whether to route and which path to use for the virtual circuit must be made without access to the current network state. Second, a router's image of the network state can change only after a request is made to that router. Without these conditions, the problem would reduce to a single router problem.

In our lower bounds proofs, we will assume that a router obtains updated information about the state of the entire network after it handles a request. Our algorithms (upper bounds) only require the information which could be obtained by "piggybacking;" in other words, the loads along the edges where the request is routed.

An algorithm for this problem must guarantee that for every edge e and time τ, the total number of requests on e which are active at time τ does not exceed u_e. When the request sequence terminates, our algorithm receives total profit $\rho = \sum_{i \in R} p_i$ where R is the set of routed requests. We define ρ^* to be the maximum possible profit attainable for the request sequence without overflowing capacity (i.e. optimum profit). Our competitive ratio is the maximum over request sequences of ρ^*/ρ.

4 Distributed Exponential-Based Routing

We present a deterministic algorithm for the Multiple Router problem which attains $O(\log \mu)$ competitiveness with k routers, provided the capacities in the graph are at least $\Omega(k \log \mu)$. This algorithm need only modify the network image when requests are accepted, and only updates the perceived loads along the path where the request was routed. While these properties are not required by the model, they make the algorithm more realistic to implement for a large network.

We prove a matching bound on the tradeoff between competitive ratio and capacity, showing that we need randomness in order to design $O(\log \mu)$ competitive algorithms without requiring $u_{min} \geq \Omega(k \log p_{max}\tau_{max})$ and $u_{max} \geq \Omega(k \log \mu)$.

Randomized algorithms can obtain $O(\log \mu)$ competitivity while only requiring $u_{min} \geq \Omega(k + \log \mu)$. We present a randomized algorithm which matches this bound in Appendix B. While the algorithm itself is not particularly practical, we observe that the implications for networks with nonadversarial traffic are promising.

4.1 Deterministic Exponential-Based Routing

Algorithm: We will run the AAP (Awerbuch, Azar, Plotkin) algorithm [5], modifying our analysis to account for out-of-date information. Each router r will keep its own image of the loads (λ) on every edge. We will require that $u_e \geq k + k \log \mu$ for every edge. The algorithm works as follows:

1. Initialization:
   - Set the initial network loads to zero, $\lambda_e(\tau) = 0$ for all e and τ.
   - Set up each router's network image, $\lambda_e^r(\tau) = 0$ for all e, r, τ.
2. When request $i = (s_i, T_i, \tau_i^s, \tau_i^f, p_i, r_i)$ arrives:
   - Compute cost $c_e^{r_i}(\tau) = \sum_{\tau=\tau_i^s}^{\tau_i^f} (\mu^{\lambda_e^{r_i}(\tau)} - 1)$ on each edge.
   - Find the min-cost path P_i connecting s_i to some sink $t_i \in T_i$.
   - If the cost of P_i is at least p_i, then reject the request.
   - Otherwise route the request along path P_i and increase the network loads $\lambda_e(\tau)$ by $1/(u_e - k)$ for all $e \in P_i$ and τ between τ_i^s and τ_i^f.
   - If we routed the request, then for all $e \in P_i$ and τ in the duration, set $\lambda_e^{r_i}(\tau) = \lambda_e(\tau)$, updating the router's network image. Notice that the image for other routers is unchanged, and we only perform this update on edges in P_i and only when we route.

4.2 Analysis

We define $\lambda_e(\tau, i)$ to be the network load on edge e at time τ immediately after request i was handled. Similarly, $\lambda_e^r(\tau, i)$ represents the network image for router r on edge e and time τ immediately after request i was handled. In addition, $c_e(\tau, i)$ represents the cost after request i and $c_e^r(\tau, i)$ is the cost according to router r after request i. Define L to be the total number of requests.

We need to prove two things. First, we must show that our algorithm doesn't overflow capacity constraints. Second, we need to show that the total profit of routed requests is competitive against the optimal profit.

Lemma 4.1 *For all edges e and times τ, $\lambda_e(\tau, L) \leq 1 + k/(u_e - k)$.*

Proof: Suppose a router r has $\lambda_e^r(\tau, i) \geq 1$. Then we have $c_e^r(\tau, i) \geq \mu - 1$. In order to route request i on edge e, we would have to have $\mu - 1 < p_i$, which is impossible by the definition of μ (assuming at least two nodes in the network, $\mu \geq 2p_{max} \geq p_i + 1$). So no router can route on an edge after its perceived load reaches one. Each router can thus route on a load one edge at most once, adding $k/(u_e - k)$ to the load. ■

Theorem 4.1 *The algorithm never overflows the capacity of any edge at any timestep.*

Proof: The total number of requests carried by an edge at time τ is given by $(u_e - k)\lambda_e(\tau, L)$. Using lemma 4.1, this means edge e cannot carry more than u_e requests at time τ. ■

The total profit of our algorithm will be the sum of profits of routed requests. We will first prove a lemma relating total profit to total cost of edges, then use the lemma to prove our competitive ratio.

Lemma 4.2 *If we define ρ to be the total profit of the online algorithm then:*

$$\sum_\tau \sum_e (u_e - k)c_e(\tau, L) < 12(\log \mu)\rho$$

Proof: The proof is similar to one appearing in [5] and we will defer it to Appendix A. ■

Theorem 4.2 *Our competitive ratio is bounded above by $O(\log \mu)$.*

Proof: Define Q to be the set of requests which the offline algorithm accepts but the online does not. The total profit of these requests will be denoted by ρ_Q. Clearly $\rho^* \leq \rho_Q + \rho$. Consider a request j in Q. Since the online algorithm did not route j, we know:

$$p_j < \sum_\tau \sum_e c_e^{r_j}(\tau, j) \leq \sum_\tau \sum_e c_e(\tau, L)$$

Summing over all the j in Q and keeping in mind that the offline algorithm may route at most u_e requests on edge e, we get:

$$\rho_Q \leq \sum_\tau \sum_e (u_e) c_e(\tau, L) \leq \sum_\tau \sum_e \frac{u_e}{u_e - k}(u_e - k)c_e(\tau, L)$$

Using the capacity bound of $u_e \geq k + k \log \mu$ and the lemma, this means:

$$\rho_Q \leq (1 + \frac{1}{\log \mu})12(\log \mu)\rho$$

Because ρ^* is at most the sum of ρ and ρ_Q, this implies:

$$\rho^*/\rho \leq 12(\log \mu) + 13 \leq O(\log \mu)$$

■

4.3 Randomized Exponential-Based Routing

We present a randomized algorithm which obtains $O(\log \mu)$ competitivity while assuming a granularity of only $O(k + \log \mu)$. While the algorithm itself is not very practical, we believe that the implications for deterministic approaches against non-adversarial traffic are promising. The proof of competitivity is a fairly straightforward exercise in application of Chernoff bounds to the deterministic analysis; it is left for Appendix B.

We assume that every edge has capacity at least $u_e \geq 2k + 12 \log \mu$. The randomized algorithm will pretend each link has a capacity of $u_e^V = (4 \log 2\mu) u_e$. We will run the deterministic algorithm on this virtual network. Each time the deterministic algorithm rejects, the randomized algorithm also rejects. If the deterministic algorithm accepts, then the randomized algorithm will only accept with a fixed probability $\Pi = 1/(8 \log 2\mu)$. The details of the algorithm follow.

1. Initialization:
   - Set the initial network loads to zero, $\lambda_e(\tau) = 0$ for all e and τ.
   - Set the initial virtual network loads to zero, $l_e(\tau) = 0$ for all e, τ.
   - Set up each router's network image, $\lambda_e^r(\tau) = 0$ for all e, r, τ.
   - Set up each router's virtual image, $l_e^r(\tau) = 0$ for all $e, r, , \tau$.
2. When request $i = (s_i, T_i, \tau_i^s, \tau_i^f, p_i, r_i)$ arrives:
   - Compute virtual cost $c_e^{r_i}(\tau) = \sum_{\tau=\tau_i^s}^{\tau_i^f} \mu^{l_e^{r_i}(\tau)} - 1$ on each edge.
   - Find the min-cost path P_i connecting s_i to a sink.
   - If the cost of P_i is at least p_i, then reject the request. This is a "virtual reject."
   - Otherwise, check to see if $\lambda_e^{r_i}(\tau) > 1$ for any edge $e \in P_i$ and time τ between τ_i^s and τ_i^f. If so, we must reject to avoid overflowing capacities. This is a "capacity reject."
   - If we have not yet chosen to reject the request, we will accept with probability Π along path P_i. When we accept a request, we must increase $\lambda_e(\tau)$ by $1/(u_e - k)$ for all edges $e \in P_i$ and times τ between start and finish times. We also set $\lambda_e^{r_i}(\tau) = \lambda_e(\tau)$ along the path.
   - If we did not choose to "virtual reject" (even if the actual algorithm rejected), we update the virtual loads, increasing $l_e(\tau)$ by $1/(u_e^V - k)$ and setting $l_e^{r_i}(\tau) = l_e(\tau)$ for all $e \in P_i$ and τ between start and finish times.

5 Lower Bounds for Online Distributed Routing

We will show various capacity lower bounds on the graph which are necessary for a deterministic algorithm to attain $O(\log \mu)$ competitivity. Specifically, we show that every edge must have capacity at least $u_{min} \geq \Omega(k \log p_{max})$ and $u_{min} \geq \Omega((k \log \tau_{max})/(\log\log \tau_{max}))$, and that some edge must have capacity at least $u_{max} \geq \Omega((k \log n)/(\log \log n))$.

Theorem 5.1 *For any deterministic algorithm, there exists a graph and request sequence such that the algorithm cannot achieve $O(\log p_{max})$ competivity unless $u_{min} \geq \Omega(k \log p_{max})$.*

Proof: The graph will be a single edge of capacity u_e. The adversarial request sequence will consist of several phases of requests along this edge, where all requests made in the same phase have identical profit value. If we define γ to be the competitive ratio of the online algorithm, then all requests in phase i will have profit value

$$p_i = [\frac{\gamma k}{u_e} + 1]^i$$

The adversary's request sequence terminates when $p_i > p_{max}$. There are thus $\log p_{max} / \log((\gamma k/u_e) + 1)$ stages.

During phase i, the adversary first "primes" the routers. We give each router requests of profit p_i until the next request would be accepted. Lemma 5.1 will show that this "priming" is possible and does not require more than u_e requests to perform. Once the priming is complete, the request sequence continues with one more profit p_i request to each router. Since the routers' network images haven't changed since they were primed, each router will accept. The algorithm accepts exactly k requests in every stage.

Lemma 5.1 *The adversary can prime the routers for phase i using at most u_e requests.*

Proof: Suppose the adversary can't prime the routers using u_e requests. Then the adversary could give out u_e requests without any router accepting. As we are in phase i, the total profit of the online algorithm is:

$$\rho = \sum_{j=0}^{i-1} k p_j = \sum_{j=0}^{i-1} k[\frac{\gamma k}{u_e} + 1]^j = k([\frac{\gamma k}{u_e} + 1]^i - 1)/[\frac{\gamma k}{u_e}] = \frac{(p_i - 1)u_e}{\gamma}$$

However, the optimal profit involves accepting the u_e requests of profit p_i which we just rejected. This yields an optimal profit of:

$$\rho^* = (u_e)p_i$$

The competitive ratio would then be

$$\rho^*/\rho = \gamma \frac{p_i}{p_i - 1} > \gamma$$

.

This contradicts the definition of γ, so the lemma must hold. ■

When all phases complete, the algorithm must not have exceeded the edge capacity. We use this fact to complete the proof of the theorem.

The adversary continues until $p_i > p_{max}$. This takes a total of $i = (\log p_{max})/\log((\gamma k/u_e) + 1)$ phases. In each phase, the algorithm must accept k requests. So in order to keep from overflowing capacity bounds, we require $ki \leq u_e$. This implies:

$$\frac{k \log p_{max}}{u_e} \leq \log(\frac{\gamma k}{u_e} + 1)$$

Exponentiating both sides gives the inequality:

$$\frac{\gamma k}{u_e} + 1 \geq 2^{(\log p_{max})k/u_e}$$

.

If $\gamma = \log p_{max}$, this is of the form $x + 1 \geq 2^x$ and can only hold for $\log p_{max} = \gamma \leq u_e/k$, proving that in order for $\gamma = O(\log p_{max})$ to hold, we require $u_e \geq \Omega(k \log p_{max})$.

In a network with multiple edges, we can create the same situation by giving requests which must traverse a particular edge; thus every edge must have $u_e \geq \Omega(k \log p_{max})$. ■

A similar argument shows that any line with at least n nodes must contain some edge with capacity at least $\Omega((k \log n)/(\log \log n))$.

Theorem 5.2 *For any deterministic algorithm, there exists a graph and request sequence such that the algorithm cannot achieve $O(\log n)$ competitivity unless some edge has capacity at least*

$$u_{max} \geq \Omega((k \log n)/(\log \log n))$$

Proof: This time, the graph will be a line of n edges. The adversary will again proceed in phases. In the first phase, the adversary requests connections from the left end of the line to the right end. It first primes each router so the next request will be accepted, just as before. Once k requests have been accepted, one by each router, the adversary splits the line into $s = \gamma + 1$ subsections. Each subsection is assigned an equal portion of the profit so far. The adversary then recurses on each subsection. If at any time the adversary is unable to prime a subsection using u_{max} requests, the adversary halts on that subsection. In any case, the adversary halts after $\log_s n$ phases, when the subsections reach length one.

Lemma 5.2 *If the adversary halts on a subsection of length $L > 1$, then the competitive ratio on that subsection is worse than γ.*

Proof: In order to get a subsection of length L, the algorithm must have run $f = (\log(n/L))/(\log s)$ phases. In each phase, k requests were accepted along the subsection in question, and in each phase the assigned profit per subsection was divided. The total profit the online algorithm assigns to the subsection is:

$$k \sum_{i=1}^{f} (\frac{1}{s})^i = ks^{-f} \sum_{i=0}^{f-1} s^i = ks^{-f} \frac{s^f - 1}{s - 1}$$

The optimal profit is at least equal to the minimum edge capacity, obtained by filling the subsection with short requests. This capacity is always at least k (see theorem 5.1 and theorem 5.4). Dividing optimal by online gives a competitive ratio of:

$$\frac{k(s-1)s^f}{k(s^f-1)} \geq \gamma \frac{s^f}{s^f-1} > \gamma$$

. ■

The total online profit is the sum over subsections of the online profit. The optimum profit is similarly the sum over subsections of the optimum profit. It follows that in order to attain an overall competitive ratio of γ, some subsection must have competitive ratio at least γ. Using lemma 5.2, this requires the subsection to have length one when the adversary terminates; therefore that subsection completed all $(\log n)/(\log s)$ phases. The capacity on this subsection must be at least $k(\log n)/(\log \gamma + 1)$. If $\gamma = O(\log n)$ it follows that $u_{max} \geq \Omega((k \log n)/(\log \log n))$. ■

Theorem 5.3 *For any deterministic algorithm, there exists a graph and request sequence such that the algorithm cannot achieve $O(\log \tau_{max})$ competitivity unless every edge has capacity at least*

$$u_{min} \geq \Omega((k \log \tau_{max})/(\log \log \tau_{max}))$$

.

Proof: By examining the time-dilation of the graph, a single edge viewed over τ_{max} becomes a line of length τ_{max}. Theorem 5.2 applies and it follows that each edge must, at some time step, have capacity at least $\Omega((k \log \tau_{max})/(\log \log \tau_{max}))$. Since edge capacity does not vary with time, it follows that each edge must have at least the desired capacity. ■

The above lower bounds apply only for deterministic algorithms. In the randomized case, the work of [5] gives an $O(\log \mu / \log \log \mu)$ bound on the necessary granularity. We present an $O(k)$ bound.

Theorem 5.4 *We require at least k capacity on each link to guarantee that there will be no overflows.*

Proof: Suppose some edge has capacity less than k. Suppose some router has probability less than $1/\gamma$ to accept a request on this edge. We could give that router a request and then stop, giving expected competitive ratio at least γ. Assuming some finite competitive ratio, every router has a nonzero probability of accepting. It follows that if we give one request along the link to every router, there is a nonzero probability that every router accepts, causing the edge to overflow. ■

References

[1] S. Albers, M. Charikar, and M. Mitzenmacher. Delayed information and action in on-line algorithms. *39th IEEE symposium on Foundations of Computer Science*, pages 71–80, 1998.

[2] 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.

[3] A. Awerbuch and Y. Azar. Local optimization of global objectives: competitive distributed deadlock resolution and resource allocation. *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 240–49, 1994.

[4] A. Awerbuch and Y. Azar. Competitive multicast routing. *Wireless networks*, 1(1):107–14, 1995.

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

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

[7] X. Deng and C.H. Papadimitriou. On the value of information. *Proceedings of the 12th IFIPS Congress, Madrid*, 1992.

[8] J. Garay, I. Gopal, S. Kutten, Y. Mansour, and M. Yung. Efficient on-line call control algorithms. *2nd Annual Israel Conference on Theory of Computing and Systems*, 1993.

[9] A. Goel, M. Henzinger, and S. Plotkin. Online throughput-competitive algorithm for multicast routing and admission control. *9th ACM-SIAM Symposium on Discrete Algorithms*, 1998.

[10] A.L. Gupta and N.D. Georganas. Analysis of a packet switch with input and output buffers and speed constraints. *IEEE Infocom*, pages 694–700, 1991.

[11] S. Irani and Y. Rabani. On the value of information in coordination games. *34th IEEE symposium on Foundations of Computer Science*, pages 12–21, 1993.

[12] C.H. Papadimitriou and M. Yannakakis. Linear programming without the matrix. *Proceedings of the 25th ACM Symposium on Theory of Computing*, pages 121–29, 1993.

A Proof of Lemma 4.2

The lemma states that, if ρ is the total profit of the online deterministic algorithm given in Section 4.1, then the cost function after the request sequence termincates satisfies the property:

$$\sum_{\tau} \sum_{e} (u_e - k) c_e(\tau, L) < 12(\log \mu)\rho$$

Proof: Summing over requests, we get the following:

$$\sum_{\tau} \sum_{e} (u_e - k) c_e(\tau, L) = \sum_{j} \sum_{\tau} \sum_{e} (u_e - k)(c_e(\tau, j+1) - c_e(\tau, j))$$

The cost function changes on (e, τ) only when a circuit which is active during time τ is routed along a path including e. When this happens, the load is increased by exactly $1/(u_e - k)$. This implies the following inequality (where R is the set of routed requests):

$$\sum_j \sum_\tau \sum_e (u_e - k)(c_e(\tau, j+1) - c_e(\tau, j)) \le \sum_{j \in R} \sum_{\tau=\tau_j^s}^{\tau_j^f} \sum_{e \in P_j} (u_e - k) c_e(\tau, j)(\mu^{1/(u_e-k)} - 1)$$

Substituting $2^{\log \mu}$ for μ yields:

$$\sum_\tau \sum_e (u_e - k) c_e(\tau, L) \le \sum_{j \in R} \sum_{\tau=\tau_j^s}^{\tau_j^f} \sum_{e \in P_j} (u_e - k)(2^{(\log \mu)/(u_e-k)} - 1) c_e(\tau, j)$$

By the assumed capacity bound, we know that $\log \mu \le u_e - k$, so using the fact that $2^x - 1 \le x$ for $0 \le x \le 1$, we can write:

$$\sum_\tau \sum_e (u_e - k) c_e(\tau, L) \le (\log \mu) \sum_{j \in R} \sum_{\tau=\tau_j^s}^{\tau_j^f} \sum_{e \in P_j} c_e(\tau, j)$$

We will consider two separate cases. In the "good" case, we require $\lambda_e^{r_j}(\tau, j) + (2k/(u_e - k)) \ge \lambda_e(\tau, j)$. This implies, in the "good" case:

$$c_e^{r_j}(\tau, j) \le c_e(\tau, j) \le \mu^{2k/(u_e-k)} c_e^{r_j}(\tau, j) \le \mu^{2/(\log \mu)} c_e^{r_j}(\tau, j) \le 4 c_e^{r_j}(\tau, j)$$

In the "bad" case, at least $2k$ requests were routed over the current edge and time unit since our router last examined the load. Of the first $2k$ requests routed on the edge and time unit, all must be in the "good" case. Consider $2k$ consecutive routings on the edge and time unit. No single router may have more than one "bad" routing in this span. Since there are k routers, this means that at least k "good" routings occurred in any sequence of $2k$. We may pair off "bad" routings with "good" ones, with at most k routings occurring between members of the pairing. This means the total cost when "bad" routings occur is bounded by $\mu^{k/(u_e-k)} \le 2$ times the total cost when "good" routings occur.

From this, we conclude:

$$(\log \mu) \sum_{j \in R} \sum_{\tau=\tau_j^s}^{\tau_j^f} \sum_{e \in P_j} c_e(\tau, j) \le 3(\log \mu) \sum_{j \in R} \sum_{good(\tau,e)} c_e(\tau, j) \le 12(\log \mu) \sum_{j \in R} \sum_{\tau=\tau_j^s}^{\tau_j^f} \sum_{e \in P_j} c_e^{r_j}(\tau, j)$$

Since we're summing over requests that were actually routed, we know:

$$\sum_{\tau=\tau_j^s}^{\tau_j^f} \sum_{e \in P_j} c_e^{r_j}(\tau, j) < p_j$$

Substituting this, and noticing that the sum of p_j over requests routed is equal to ρ, we have shown that:

$$\sum_\tau \sum_e (u_e - k) c_e(\tau, L) < 12(\log \mu)\rho$$

■

B Analysis of the Randomized Algorithm

We again represent the cost after request i by $c_e(\tau, i)$, and define L to be the final request. We will first prove that the deterministic virtual algorithm obtains within a constant factor of the optimal profit.

Lemma B.1 $\rho^V \geq \rho^*/5$

Proof: From lemma 4.2 on the deterministic algorithm, we know:

$$\sum_\tau \sum_e (u_e^V - k) c_e(\tau, L) \leq 12(\log \mu)\rho^V$$

By repeating the steps in the proof of competitiveness for the deterministic algorithm (theorem 4.2), we can show that if ρ_Q is the total profit of requests accepted by the optimal algorithm but rejected by the deterministic virtual algorithm, then:

$$\rho_Q < \sum_\tau \sum_e u_e c_e(\tau, L)$$

Substituting, and keeping in mind that $u_e^V = (4 \log 2\mu) u_e$:

$$\rho^* \leq \rho^V + \rho_Q \leq \rho^V + 12(\log \mu)(\frac{u_e}{u_e^V - k})\rho^V \leq 5\rho^V$$

■

We need to show that capacity rejects are sufficiently rare that the competitive ratio looks like $O(\Pi)$. We will first bound the chance of filling up a single edge, then use this to prove the desired results.

Lemma B.2 *For any given e and τ, $Pr[\lambda_e(\tau) \geq 1] \leq 1/\mu^{2.3}$.*

Proof: The deterministic virtual algorithm accepts at most u_e^V requests along edge e at time τ. Each of these requests has probability at most Π of being accepted by the real algorithm. This holds regardless of changes to the request sequence by the adaptive adversary; Π is a fixed constant independent of the previous choices of the algorithm. The load is thus bounded by a sum of Bernoulli variables. The expected number of requests on e is $u_e^V/8\log\mu = u_e/2$, for expected load 1/2. Applying Chernoff bounds:

$$Pr[\lambda_e(\tau) \geq 1] < (\frac{e}{4})^{u_e/2} \leq (\frac{e}{4})^{6 \log \mu} < \mu^{6 - 6\log 4} < 1/\mu^{2.3}$$

■

Lemma B.3 *The probability of at least one capacity reject occuring on requests with start time τ is at most $1/\mu^{0.3}$.*

Proof: Every request with start time τ finishes by $\tau + \tau_{max}$. In order for a capacity reject to occur, there must be some edge-time-router triple such that $\lambda_e^r(\tau) \geq 1$. Since each router's image has $\lambda_e^r(\tau) \leq \lambda_e(\tau)$ it follows that $\lambda_e(\tau) \geq 1$. For any given edge and time the probability of this is at most $1/\mu^{2.3}$ by lemma B.2. The chance of this happening for one of the n^2 edges at one of the τ_{max} times between τ and $\tau + \tau_{max}$ is at most $n^2\tau_{max}/\mu^{2.3} \leq 1/\mu^{0.3}$. ■

We now prove the expected competitive ratio. Notice that the future actions of the algorithm are in no way effected by the previous coin tosses (unless a capacity reject occurs). Since there is no way the adaptive adversary can increase the chance of a capacity reject above $1/\mu$ on any time interval of length τ_{max} it will follow that we are competitive against an adaptive adversary.

Theorem B.1 *The expected competitive ratio of the algorithm is $O(\log \mu)$.*

Proof: We define ρ_τ^V to be the total profit obtained by the virtual deterministic algorithm on requests with start time τ. Similarly, ρ_τ is our algorithm's profit on requests with start time τ. Using lemma B.1 we can write:

$$\rho^* \leq 5\rho^V = 5\sum_\tau \rho_\tau^V$$

Each of the requests which the virtual deterministic algorithm accepted has a probability Π of being accepted by the actual algorithm, unless a capacity reject occurs. The times when a capacity reject occurs reduce the profit by at most ρ_τ^V and happen with probability at most $1/\mu^{0.3}$ according to lemma B.3. We can write:

$$E[\rho_\tau] \geq (\Pi)\rho_\tau^V - \frac{\rho_\tau^V}{\mu^{0.3}}$$

Combining equations and using linearity of expectations, we can write the following:

$$E[\rho] = \sum_\tau E[\rho_\tau] \geq (\Pi - \frac{1}{\mu^{0.3}})\sum_\tau \rho_\tau^V \geq \frac{\rho^*}{5}(\Pi - \frac{1}{\mu^{0.3}}) = \frac{\rho^*}{5}(\frac{1}{8\log\mu} - \frac{1}{\mu^{0.3}}) = \frac{\rho^*}{O(\log\mu)}$$

■