Efficient Topology Update on High Speed Wide Area Networks

by

Matthias David Siebler

Submitted to the Department of Electrical Engineering and Computer Science

in partial fulfillment of the requirements for the degrees of

Master of Engineering in Electrical Engineering and Computer Science

and

Bachelor of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1995

Author .... [illegible]
Department of Electrical Engineering and Computer Science
May 21, 1995

Certified by .... [illegible]
Baruch Awerbuch
Research Scientist
Thesis Supervisor

Accepted by .... [illegible]
F.R. Morgenthaler
Chairman, Department Committee on Graduate Theses

Efficient Topology Update on High Speed Wide Area Networks

by

Matthias David Siebler

Submitted to the Department of Electrical Engineering and Computer Science
on May 21, 1995, in partial fulfillment of the
requirements for the degrees of
Master of Engineering in Electrical Engineering and Computer Science
and
Bachelor of Science in Computer Science and Engineering

Abstract

Distributed routing and call admission algorithms for high speed wide area networks typically rely on link state information for optimal decision making. Therefore, topology updates are necessary to have accurate local network utilization tables. The update strategy must be efficient enough to avoid taxing the resources of the network while maintaining the performance of the algorithms. Performance is generally characterized as the amount of traffic which can be admitted into the network. Using both theoretical and empirical techniques, this thesis studies the effect of inaccurate local databases and the amount of information necessary to achieve good performance.

Thesis Supervisor: Baruch Awerbuch

Title: Research Scientist

Acknowledgements

First of all, I would like to thank my parents for their unconditional love and unwavering support for these last five years. Words cannot express my deep respect and gratitude.

I would like to thank my advisor, Baruch Awerbuch, for guiding and supporting me, both as an undergraduate researcher and during my graduate work.

Rainer Gawlick deserves to be recognized for all his advice and encouragement, which went far beyond the call of duty. Without him, this thesis would not have been possible.

Finally, I would like to thank all my brothers in Kappa Sigma, who made my college years such a incredible experience. A.E.K.Δ.B.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The advent of fiber optic technology has created a new class of wide area networks. These high speed networks, such as PARIS/plaNET [2, 6] will have bandwidths in the Gigabit rate range and will support large numbers of users. Along with offering new capabilities, these networks also create new problems. One such problem is how to maintain distributed network state information.

This research project aims to study the problem of efficiently maintaining topology and network utilization information. The project has two parts, one focusing on analytically determining the effect of incomplete knowledge on a distributed routing and call admission algorithm. The second part simulates the effect of partial state information on the performance of the algorithm. It is hoped that this project will shed new light on this area.

1.1 Background and Motivation

Managing a limited resource such as bandwidth is a problem common to most networks. This problem is handled by algorithms which decide how much traffic should be admitted to the network and along which path to send it to its destination, respectively denoted the *call admission* and *routing* problems. Routing and call admission algorithms require a certain amount of information about the current state of the network. In a real-time environment, this information can never be completely accurate. The *topology update* question is then how often and in which manner this information must be improved. A large body of research addresses the problems of managing bandwidth and topology updates [8]. For example, Amotz Bar-Noy and Madan Gopal [5] examined five different update strategies and their effects on a distributed routing algorithm. Their research focused on the problem of finding shortest paths between networks, given that local hosts have only limited knowledge about the other networks.

Routing, call admission and topology update have been studied extensively for many years. Much of the current research on these problems has been spurred by the need for integrated service networks. Integrated service combines multiple types of data, such as voice, video and file transfer into a single network. These networks require a protocol such as Asynchronous Transfer Mode (ATM) which has the dual advantages of low delay and flexibility [10, 9, 13]. The low delay comes from the use of circuit routing, which allows fast packet switching, while the flexibility stems from the ability to multiplex several connections together. Since cells in ATM are source routed, resources can be reserved to meet various Quality of Service (QoS) requirements. This protocol was designed to be flexible enough to effectively manage multiple types of data.

Fiber optic wide area networks are designed to service large numbers of users, each expecting a high quality of service. The demand is sufficiently large enough to render centralized control infeasible. Decisions must be made immediately as each request arrives without any knowledge of future demand. To further complicate the situation, each node may only have partial information about the current state of the network, since connections may simultaneously arrive at other nodes. The algorithm must be efficient enough to prevent admission from becoming a bottleneck. Similarly, after acceptance of a connection, frequent rerouting becomes impractical, since ensuring uninterrupted service makes rerouting very expensive.

These complications are what led to the consideration of distributed routing algorithms. *Distributed* routing differs from *centralized* control in that decision making occurs locally instead of at a global network controller. The performance of high speed networks would be severely compromised if each host had to depend on a remote centralized node for decision making. Distributed algorithms also have the advantage of greater reliability and security, since operation of the network does not depend on a single location.

While distributed algorithms may possess these advantages, they induce additional complexity. For efficient management of the network, each individual node must be capable of informed decisions. Some sort of topology update mechanism is therefore required. Previous research [2, 7] has shown the feasibility of maintaining structures for network broadcast. This allows any node to quickly inform the network of the load on its adjacent links. The question remains of how frequently these updates should be sent. Obviously, sending them too frequently will clog the network with needless messages. The local hosts will be overwhelmed if too many updates are broadcast, which is conceivable considering the size of some networks. On the other hand, if updates are too infrequent, routing decisions will suffer from a lack of current information. The goal is to find the right balance.

The motivation for examining new distributed algorithms is that most of the existing research focuses on packet routing. Protocols which use packet routing such as TCP/IP cannot effectively handle integrated service. A packet routed network cannot reserve bandwidth and therefore cannot make any guarantees about the performance of the network. The routing algorithms designed for these protocols are therefore not very useful in an ATM environment. This environment requires new solutions.

An important consideration in designing new algorithms for ATM networks is how the three problems of routing, admission control and topology update interact. The algorithms must be designed to complement each other. Bahk and El Zarki [4] designed a dynamic multi-path algorithm which attempted to bridge the gap between routing and admission control. Their approach is to use shortest-path routing if traffic is light, but then switch to multi-path routing once the network load increases. A hierarchical approach, proposed by Meempat and Sundareshan [12], would use a network supervisor to control access. The algorithm uses centralized admission control but distributed routing. Lin and Yee [11] present a completely distributed routing and admission control algorithm for use in ATM networks. Their algorithm formulates the problem as a combinatorial optimization problem, and then attempts to use Lagrangian relaxation to obtain a good solution.

Of the above researchers, only Bahk and El Zarki addressed the question of how link state information is updated. They proposed an estimation scheme which computes an effective capacity for each link. This estimate is updated by infrequent broadcasts from each network node. However, none of the other researchers addressed the problem of how often the link state information which all their algorithms depend upon will be updated. Most of the research only looks at the interaction of routing and admission control. This paper differs from previous research by examining the interaction of all three problems. The aim is to discover how much information is actually necessary for good algorithm performance. This paper uses an algorithm developed by Awerbuch, Azar and Plotkin [1] which solves the call admission and routing problems. By combining this algorithm with a simple topology update mechanism, it is hoped to understand how the lack of accurate link state information affects the system.

1.2 Description and Goals

How frequently should utilization updates be broadcast? This is the question that will be addressed by the experimental component of this project. Specifically, a distributed algorithm will be implemented on a network simulator. The performance of this algorithm under varying conditions will then be analyzed to determine the optimal topology update strategy. Two different topologies will be used. The networks will be simulated under several degrees of usage. For each type of workload, the frequency of topology updates can be adjusted to balance the need for fresh information against the stress on the local host computation from too many messages. Typical qualitative measures such as mean throughput and *loss ratio* will be considered. Loss ratio is the percentage of rejected connections out of the total workload. These measures form a picture of how the network is behaving.

The distributed algorithm considered solves the call admission and routing problems in a *virtual circuit* environment. In this case, all packets for a connection use the same route through the net. For a network with many users and varied data types, including real-time audio and video, circuit routing is a method of guaranteeing minimum latency and reserving bandwidth. As mentioned earlier, circuit routing is necessary because packet routing cannot reserve bandwidth. This inability of a packet routed network to provide flexible QoS guarantees, which is critical for multimedia applications, stems from the fact that every packet can take a different route through the network. Routing in virtual circuits assumes that each connection has a certain bandwidth demand which must be satisfied for the duration of the call. Once a route has been established, rerouting or interrupting it is very costly, due to the large amount of data frames which would accumulate at a node's buffer. As a result, paths are static for a call's duration. Also, blocked calls do not reapply for admission into the system. These constraints apply to every algorithm.

Now that the constraints are established, the two schemes used for comparison can be explained. Baruch Awerbuch, Yossi Azar and Serge Plotkin developed the *Route_or_Block* protocol [1], an effective combined routing and call admission algorithm that has the advantage of being relatively easy to implement. This algorithm chooses a routing path for a connection and regulates the entrance of heavy users into the system. To work correctly, this algorithm requires each network host to have knowledge of the system's current state. As a control, the minimum-hop heuristic is being used. Both the *Route_or_Block* algorithm and min-hop are distributed. Brief descriptions of each follow.

Route_or_Block accepts or rejects a connection request immediately based on a cost versus value comparison. This algorithm makes its decisions *on-line*, without any knowledge of future arrivals, in contrast to *off-line* algorithms, which know the entire input sequence of connection requests. When a call arrives, the local controller looks for the lowest cost path that can accommodate that demand. The *cost* of a path is a function of its utilization. If this cost is greater than the profit or *value* of the request, it is rejected. The value of a request can depend on the quantity of information it will transmit. Otherwise, the cheapest path is assigned to that connection. Minimum-hop simply chooses the shortest path between the source and destination, which is a static method of routing.

The previous paragraphs explained the empirical portion of this thesis. The theoretical part considers the effect of partial state information on the *competitive ratio* of the algorithm. Consider the ratio of the worst-case performance by an on-line algorithm against the optimum off-line algorithm on the same sequence of inputs. The competitive ratio is the maximum value of this ratio over all inputs [3]. Since a certain amount of guessing is involved, any on-line algorithm will perform less effectively than an optimal off-line scheme. However, algorithms performing within a certain factor are efficient enough to be useful. Proving the competitive ratio also guarantees a minimum level of performance over all inputs.

This research project has multiple goals. One is to analytically study the effect of erroneous network state information on distributed routing algorithms. This question will also be addressed empirically, by using a network simulator to run a series of tests. Finally, it is hoped that this project will suggest possible avenues for future research.

This thesis is organized into the following parts. First, the effect of error is theoretically analyzed. Chapter 3 then describes the method by which the experiments were conducted. Chapter 4 explains the results of the first series of tests. The next part relates how the outcomes were effected by changing the update method. The last one discusses the results and speculates upon some possible follow up research.

Chapter 2

Theoretical Error Analysis

Algorithms can be evaluated both empirically and analytically. The goal of theoretical analysis is to prove the correctness of the algorithm and establish its competitive ratio. The algorithm used in this paper has been shown to operate correctly by Awerbuch, Azar and Plotkin [1] in their paper presenting *Route_or_Block*. In the same paper, the algorithm was also shown to be $O(\log nT)$ competitive, where n is the size of the network and T is the maximum duration of any connection request. This chapter will show how operation in an implementation with inaccurate local state information effects this analysis and competitive ratio.

Before the algorithm can be evaluated, some definitions and assumptions must be made. A network is defined as a directed graph $G(V, E)$ with a capacity function $u(e)$ defined over all edges. The input to the graph is a sequence of connection requests β_i, where each request consists of an origin-destination pair (s_i, d_i), a starting and ending time (T_i^s, T_i^f), a rate function $r_i(t)$ and a connection profit ρ_i. The rate function is defined at all times, but is equal to zero outside of the interval between the start and end times. The utilization is measured by the relative load of each edge,

$$\lambda_e(t, j) = \sum_{i<j|e\in P_i} \frac{r_i(t)}{u(e)}$$

which is the sum of the rates of all the calls using that edge admitted before β_j arrives at the host. Thus, the load is a function of the time and the requests admitted.

The relative load of an edge determines the cost of that edge. This cost is weighted by the capacity of the edge. Define $\mu = 2nTF + 1$ as the cost constant. Then the cost of an edge

$$c_e(t,j) = u(e)(\mu^{\lambda_e(t,j)} - 1)$$

is exponentially proportional to its load. For a request to be accepted, there must exist a path where the cost of the path is less than its profit. The path cost is a weighted sum of edge costs.

However, since this algorithm is distributed, the local hosts do not have access to the actual utilization and cost of each edge. Rather, each host has an estimated load for each link $l_e(t,j)$ and an estimated cost

$$x_e(t,j) = u(e)(\mu^{l_e(t,j)} - 1)$$

which is the value used for computing paths. The update mechanism of the network keeps the local estimations within a certain bound of the real values. Let τ be the update threshold, expressed as a percentage of capacity. If an edge's load has changed by that amount since its last update, the edge must broadcast its actual load throughout the network.

There are several basic assumptions made about this algorithm. The first is that since an edge will trigger an update if its load varies past the threshold, the error of the estimated load $l_e(t,j) = \lambda_e(t,j) \pm \tau$ is bounded. The second assumption normalizes the profit of each request. First define $T_j = T_j^f - T_j^s$ to be the duration of a connection and T to be the maximum duration of all requests. Since the profit is proportional to the rate and duration product of the call, it can be normalized so that

$$1 \leq \frac{\rho_j}{nT_j r_j(t)} \leq F$$

where F is some constant. The final restriction limits the incoming requested rates to be a small fraction of the minimum link capacity. This assumption for each incoming request $r_j(t) < u(e)(1/(\log \mu) - \tau)$ means that no single connection can demand to

```
ROUTE_OR_BLOCK($\beta_j$):
    if $\exists$ a path $P$ from $s$ to $d$ such that
        $$\sum_{e \in P} \sum_t \frac{r_j(t)}{u(e)} x_e(t,j) < \rho_j$$
    then route $\beta_j$ on $P$ and set
        $\forall e \in P, T_j^s \leq t \leq T_j^f : \lambda_e(t,j+1) \leftarrow \lambda_e(t,j) + \frac{r_j(t)}{u(e)}$
        $\forall e \in P, T_j^s \leq t \leq T_j^f : l_e(t,j+1) \leftarrow l_e(t,j) + \frac{r_j(t)}{u(e)}$
    else block the connection request
```

Figure 2-1: The ROUTE_OR_BLOCK Algorithm

many resources. Given the large capacities available, a connection exceeding this bound would not be very common. This bound also requires that $\tau < 1/\log \mu$.

Next the routing and flow control algorithm is defined. The *Route_or_Block* algorithm is shown in Figure 2-1. Essentially, what this algorithm does is look for a low cost path for a request when it arrives. The utilizations and costs used in the calculations are the local estimates at the host. If there exists such a path whose cost is less than its profit, the call is accepted and assigned to the path. Otherwise, it is rejected and discarded. The amount of error at that node will effect the computation of the path's cost, since the actual cost may be higher or lower than what the local host perceives. Once a request has been accepted, the link state information at that node is updated to reflect the change. However, this information is not transmitted to the rest of the network.

The rest of this chapter closely follows the outline of the proof given in the original paper [1] by Awerbuch, Azar and Plotkin. This proof involves showing both the correctness and the competitive ratio of the *Route_or_Block* algorithm. Because a version of this proof has appeared before, some details will not be fleshed out fully. It is assumed that interested readers will reference the corresponding document. However, this chapter is self contained.

The first two lemmas seek to bound the profits, beginning with lower bounding the total profit of the *Route_or_Block* algorithm. For all the following proofs, let $\mathcal{A}$ be the set of accepted connections. Also let $\mathcal{Q}$ be the set of the connections that were rejected by the on-line algorithm yet accepted by the optimal off-line algorithm, and let m be the maximum index of any connection in this set. Finally, index k is the maximum index of all incoming requests. Once the minimum profit accumulated by this algorithm is established, the bound can be compared to the maximum profit that the off-line algorithm could possibly get. The following lemma is an inductive proof.

Lemma 2-1. *With $\mathcal{A}$ and k as defined previously,*

$$2\mu^{\tau} \log \mu \sum_{j \in \mathcal{A}} \rho_j \geq \sum_t \sum_e c_e(t, k+1)$$

Proof: Start by setting $k = 0$. This base case is obviously true. The inductive step is to show that the inequality holds each time a connection is admitted into the network. When k increments, each incoming request is either rejected or accepted. If rejected, the costs and total profit are unchanged. If accepted, the costs of the links will change like this:

$$\begin{aligned}
c_e(t, j+1) - c_e(t, j) &= u(e)\left(\mu^{\lambda_e(t,j) + \frac{r_j(t)}{u(e)}} - \mu^{\lambda_e(t,j)}\right) \\
&= u(e)\mu^{\lambda_e(t,j)}\left(\mu^{\frac{r_j(t)}{u(e)}} - 1\right) \\
&= u(e)\mu^{\lambda_e(t,j)}\left(2^{\log \mu \frac{r_j(t)}{u(e)}} - 1\right)
\end{aligned}$$

Combining this equality with the assumption about rates given earlier, and using the fact that $2^x - 1 \leq x$ for $0 \leq x \leq 1$, the relation

$$c_e(t, j+1) - c_e(t, j) \leq r_j(t)\mu^{\lambda_e(t,j)} \log \mu$$

is obtained. This is how the cost of an edge at a certain time is affected by an admission. So the total change in costs due to the admission of another request is given by summing over all the edges at all times:

$$\sum_t \sum_e (c_e(t,j+1) - c_e(t,j)) \leq \log \mu \sum_t \sum_{e \in P_j} r_j(t) \mu^{\lambda_e(t,j)}$$

To prove that the increase in costs is bounded, a relation must be established between the link utilization and connection profit. The following equation is based on the definition of the algorithm, which limits the estimated cost of an accepted link:

$$\begin{aligned}
\rho_j &> \sum_t \sum_{e \in P_j} r_j(t) \frac{x_e(t,j)}{u(e)} \\
\rho_j &> \sum_t \sum_{e \in P_j} r_j(t)(\mu^{l_e(t,j)} - 1) \\
\rho_j &> \sum_t \sum_{e \in P_j} r_j(t) \left(\frac{\mu^{\lambda_e(t,j)}}{\mu^{\tau}} - 1 \right) \\
\rho_j \mu^{\tau} &> \sum_t \sum_{e \in P_j} r_j(t) \mu^{\lambda_e(t,j)} - \sum_t \sum_{e \in P_j} r_j(t) \mu^{\tau} \\
\rho_j \mu^{\tau} &> \sum_t \sum_{e \in P_j} r_j(t) \mu^{\lambda_e(t,j)} - \sum_t |P_j| r_j(t) \mu^{\tau} \\
\rho_j \mu^{\tau} &> \sum_t \sum_{e \in P_j} r_j(t) \mu^{\lambda_e(t,j)} - \rho_j \mu^{\tau} \\
2\rho_j \mu^{\tau} &> \sum_t \sum_{e \in P_j} r_j(t) \mu^{\lambda_e(t,j)}
\end{aligned}$$

Now this equation can be plugged into the inductive step for incremental cost and profit. The profit limits the cost of accepting any connection:

$$\begin{aligned}
\sum_t \sum_e (c_e(t,j+1) - c_e(t,j)) &\leq \log \mu \sum_t \sum_{e \in P_j} r_j(t) \mu^{\lambda_e(t,j)} \\
&\leq 2\mu^{\tau} \rho_j \log \mu
\end{aligned}$$

From this equation, it can be seen that the inductive step holds with each admission control and routing decision. This completes the proof of the lemma. Clearly, the profit that the algorithm can accumulate is lower bounded by the sum of the costs incurred while running the algorithm. ◇◇

A lower bound has now been established on the profit. The second step in the proof is to upper bound the profit of the off-line algorithm. This upper bound follows from the definition of path cost.

Lemma 2-2. *The profit gained by the off-line algorithm in excess of that gained by the on-line version is bounded by*

$$\sum_{j \in Q} \rho_j \leq \sum_t \sum_e (c_e(t,m)\mu^\tau + \mu^\tau - 1)$$

Proof: Consider any request $\beta_j \in Q$ which was rejected by the on-line algorithm but accepted by the off-line. Let P_j be the path over which this request was routed. Given that β_j was blocked and that the cost function is monotonic over connections,

$$\rho_j \leq \sum_t \sum_{e \in P_j} \frac{r_j(t)}{u(e)} x_e(t,m)$$

This is the limit of the costs incurred by any one connection in the set Q. The total over all the connections in the set is:

$$\begin{aligned}
\sum_{j \in Q} \rho_j &\leq \sum_{j \in Q} \sum_t \sum_{e \in P_j} \frac{r_j(t)}{u(e)} x_e(t,m) \\
&\leq \sum_t \sum_e x_e(t,m) \sum_{j \in Q | e \in P_j} \frac{r_j(t)}{u(e)} \\
&\leq \sum_t \sum_e u(e)(\mu^\tau \mu^{\lambda_e(t,m)} - 1) \\
&\leq \sum_t \sum_e (c_e(t,m)\mu^\tau + \mu^\tau - 1)
\end{aligned}$$

In the step from the second to the third equation, the last sum can be eliminated since the off-line algorithm must also satisfy the capacity constraints. Therefore, the ratio $\frac{r_j(t)}{u(e)}$ cannot exceed unity and the term can be dropped. The step from the third to the fourth equation holds because $x_e(t,j) \leq u(e)(\mu^{\lambda_e(t,j)+\tau} - 1)$, which means that $x_e(t,j) \leq c_e(t,j)\mu^\tau + \mu^\tau - 1$. This concludes the proof of the upper bound on the off-line revenue. ◇◇

At this stage, a framework has been laid down. This framework will be used to prove the new competitive ratio of the *Route_or_Block* algorithm . The proof of the theorem then is relatively simple, since the profits have been bounded and can be compared. The key to this proof is that the link costs are a lower bound of the on-line algorithm but an upper bound of the off-line algorithm.

Theorem 2-1. *The profit of the optimal off-line algorithm is never more than* $8\log(2\mu)$ *times greater than the profit generated by the on-line algorithm.*

Proof: Let the total profit accrued by the optimal off-line algorithm be Ψ. From the preceding lemma, this value can be bounded from above by:

$$\begin{aligned}
\Psi &\leq \sum_{i\in\mathcal{A}} \rho_i + \sum_{i\in\mathcal{Q}} \rho_i \\
&\leq \sum_{i\in\mathcal{A}} \rho_i + \sum_{t}\sum_{e}(c_e(t,k+1)\mu^{\tau} + \mu^{\tau} - 1) \\
&\leq \sum_{i\in\mathcal{A}} \rho_i + 2\mu^{2\tau}\log\mu \sum_{i\in\mathcal{A}} \rho_i + \sum_{t}\sum_{e}(\mu^{\tau} - 1) \\
&\leq 2\mu^{2\tau}\log(2\mu) \sum_{i\in\mathcal{A}} \rho_i + \sum_{t}\sum_{e}(\mu^{\tau} - 1)
\end{aligned}$$

The equations follow from the previously proved lemmas, as well as the fact that since β_k is the last request, then $c_e(t,k+1)$ is the final cost of each edge. The result can be further simplified by noting that $\mu^{\tau} < 2$. Using this fact, the final bound is $8\log(2\mu)\sum_{i\in\mathcal{A}}\rho_i$ plus a constant. ◇◇

This proves the new competitive ratio of the *Route_or_Block* algorithm when used in an inaccurate environment. The overall competitive ratio of $8\log(2\mu)$ compares very favorably with the original ratio of $2\log(2\mu)$, which is only a constant factor better. The algorithm is still $O(\log nT)$ competitive. Obviously, *Route_or_Block* is very robust.

The last step of the argument is to prove that the algorithm will operate correctly by not over-utilizing any edge. The algorithm is designed so that if any new call would exceed the capacity restraints of one of the edges in the lowest cost path, then that call would not be admitted.

Theorem 2-2. *For all edges $e \in E$ at all times t,*

$$\sum_{i \in \mathcal{A} | e \in P_i} r_i(t) \leq u(e)$$

Proof: Let β_j be the first connection that caused capacity constraints to be violated. Assume that edge e is the edge which becomes overloaded. Then the relative load of this edge must exceed 1, which means that $\lambda_e(t,j) > 1 - \frac{r_j(t)}{u(e)}$. Using the assumption about request rates which bounds the maximum rate at any time, and using the definition of local edge cost, which relates to the real cost,

$$\begin{aligned}
\frac{x_e(t,j)}{u(e)} &= \mu^{l_e(t,j)} - 1 \\
&\geq \mu^{\lambda_e(t,j)-\tau} - 1 \\
&> \mu^{1-\tau-\frac{r_j(t)}{u(e)}} - 1 \\
&> \frac{\mu}{\mu^\tau}\mu^{-\frac{r_j(t)}{u(e)}} - 1 \\
&> \frac{\mu}{\mu^\tau}\mu^{\tau-1/(\log\mu)} - 1 \\
&> \frac{\mu}{\mu^{1/(\log\mu)}} - 1 \\
&> nTF
\end{aligned}$$

Now take the second assumption made, which is the relation between the revenue and the requested rate and duration. Using this assumption and inserting the equation established above, it follows that

$$\begin{aligned}
r_j(t)\frac{x_e(t,j)}{u(e)} &> r_j(t)nTF \\
&> \rho_j
\end{aligned}$$

This is an obvious contradiction, so request β_j could never have been admitted. Therefore, given the assumptions restricting the input rates, the capacity constraints hold during the execution of the algorithm. ◇◇

This concludes the theoretical analysis of the *Route_or_Block* algorithm operating in an implementation without accurate link state information. The two theorems showed that with some new restrictions on the incoming requests, and the existence of some sort of update mechanism, the algorithm will perform correctly by not violating any capacity constraints, and that it also has a competitive ratio which is only a constant factor worse than the implementation with accurate state information.

Chapter 3

Experimental Environment

The theoretical model for this project is that of a synchronous connection-oriented network. All decision making takes place in discrete intervals at access nodes. The network is represented by an arbitrary graph with bidirectional high speed links. The network topology is taken from a telecommunications network.

There were some basic assumptions made in designing the simulator, which served the dual purpose of simplifying implementation and also removing environmental variables which would have obscured the results. One assumption is that all links have equivalent capacity and all nodes possess an equal amount of processing power. A somewhat unrealistic assumption is that no nodes or edges will fail. Admission control and routing decisions are made dynamically in a distributed fashion at the source nodes. There is no queueing of requests at the hosts, since all admission control decisions are immediate. Nor do rejected requests reapply for admission at a later time. Cells are routed over virtual circuits with all packets of a connection following the same path.

There are two graphs used as networks in the experiments. (See Figure 3-1.) Both have 25 host nodes, the only difference being the number of edges between them. The 'dense' graph has a total of 47 edges, while the 'sparse' graph has only 28 between the nodes. The links in the sparse graph are a subset of the edges from the dense one. All edges are bidirectional with a capacity of 155 Mbps in each direction. Each host has a private database which records each edge's available capacity.

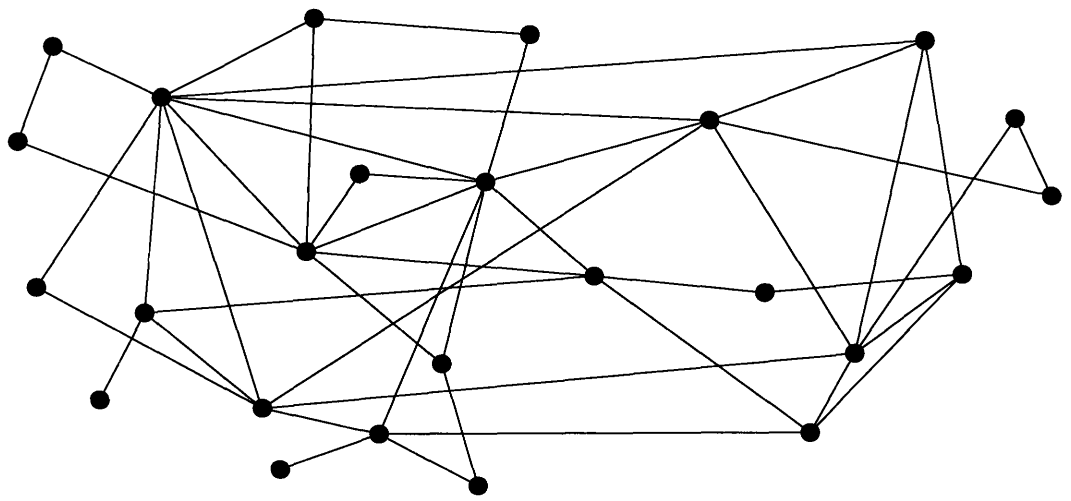
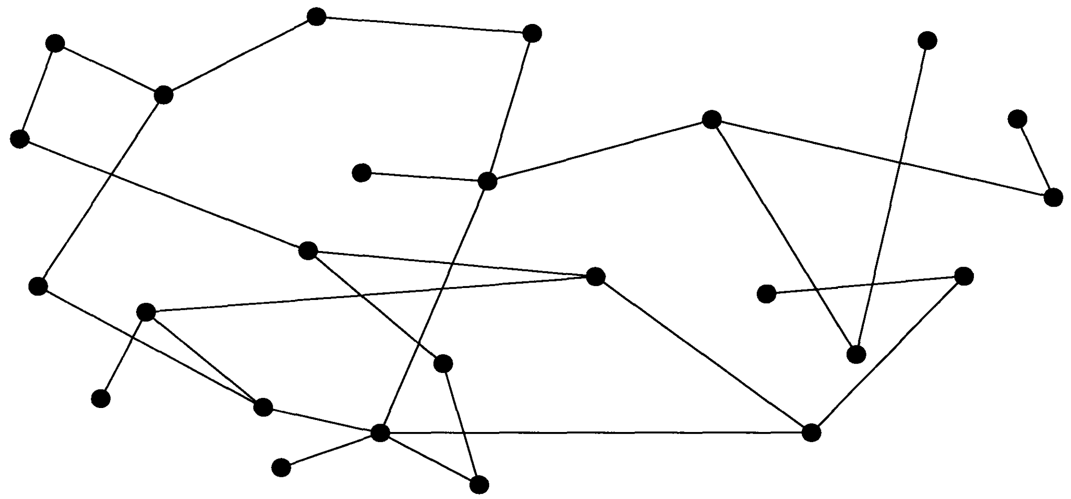

Figure 3-1: Network Topologies used for Simulations

Connection requests arrive at the network according to a Poisson process. A request consists of a source and destination host, a desired maximum rate, a duration and a starting time. The hosts are randomly chosen while the rate is a constant 1 Mbps for all connections. The duration of each request is a normal random variable with a mean of a half hour and a standard deviation of 6 minutes. The interarrival times are exponential random variables. The mean of the arrival process controls the amount of the incoming workload.

For each simulation run, requests are generated until a request arrives with a start time exceeding 100,000 seconds. Consequently, for higher interarrival rates, less traffic is presented to the system. For each request, data flows in only one direction. This is equivalent to considering a two-way connection as two separate circuits. While not all networks allow this, the assumption is not unrealistic.

The main algorithm utilized is the *Route_or_Block* routing and admission control algorithm. This algorithm works in two phases. As a request arrives at a host, the program uses Dijkstra's shortest path algorithm to determine the lowest cost path. The cost of a path is determined from the information in the source's local database. It consists of the sum of the edge weights in the path. The edge weights are an exponential function of the utilization, with a base $\mu = 100$.

Once the lowest cost path has been chosen, the second phase is to decide whether or not to admit this call. That decision is made by comparing the path cost against the request's value, which is a linear function of its bandwidth. The function used is $\rho_j = r_j(t)/5$. If the cost is too great, the request is rejected. Since these decisions are made with some degree of uncertainty, each admitted request is also checked to make sure that no edge is oversaturated. If one is, the request is rejected. Even though the algorithm has been proven to satisfy the capacity constraints, in the experimental implementation it is necessary to explicitly check to make sure this will not occur. There are a few reasons why this is necessary. First of all, the simulations do not restrict the update threshold sufficiently to ensure that no edge is overloaded. Secondly, the two constants (100 and 1/5) used by the algorithm violate some of the assumptions made in the correctness proof. The constants were chosen through empirical analysis. These compromises were accepted to ensure good experimental results.

This is then the basic algorithm. A variant is to admit all requests, routing them by the lowest cost path, regardless how high this cost is. Another option is to admit all requests along their minimum-hop path. All three algorithms will be used in the simulation runs.

The accuracy of the host databases degrade with time as the utilization of the network changes. When a connection is accepted into the network, the source node records this change and modifies its own database accordingly, decreasing the current available capacity for each edge in the chosen path. Similarly, after a connection terminates, the available capacity of the affected edges in the local database is increased by the rate of the connection. However, each node will still be unaware of any change in an edge's utilization caused by other hosts.

Topology updates are used to bring the databases up to date. An update is triggered when an edge's load changes more than the threshold value. This is a global variable that can be set to any percentage of an edge's capacity. When an edge's availability has changed more than this value since its last update, the edge notifies its source node to send an update. There are two varieties of update. Global updates are broadcast from the source node to all other nodes, while local updates reach only those hosts within a radius of two edge hops away from the source node. Both include the true available capacity of every edge emanating from the host, and are assumed to fully propagate within one time step. Multiple hosts can send updates simultaneously, but each can only send one per time step.

Multiple quantities were measured by the simulator during each run. At sampling intervals of 1000 seconds, the average edge weight, database error and network flow were recorded. Samples were not collected until after a short warmup period to ensure that the system had reached a steady state. At the end of each run, the total number of updates, the loss ratio and the average hop count were also output. Each data point is the average of five runs with different random number seeds. The confidence level was chosen to be 90%.

Chapter 4

Effects of Inaccurate Information

In this chapter, various interactions between the accuracy of local host databases and the effectiveness of the call admission and routing algorithm are explored. The goal is to discover how much information will be required at each point of entry into the network in order for the algorithm to operate effectively without consuming too many resources with operational overhead. The first section establishes the underlying correlation between error and performance, and the next one further fleshes out some interesting points. The objective was to understand the dependence of the system on the update mechanism.

4.1 Database Error

The first question examined was the relationship between database error and algorithm performance. How much will performance degrade if the local databases do not have perfect knowledge of network state? Performance in this context refers to the loss ratio, the percentage of requests that the algorithm must reject due to overloaded links, and requests which are too expensive. Loss ratio was the main performance statistic used throughout the experiments.

To answer this question, a series of tests were run at a moderate workload with varying update thresholds. The greater the update threshold, the less frequently updates would be sent and consequently, the higher the average database error. For

Dense		*Sparse*	
updates	*loss ratio*	*updates*	*loss ratio*
optimal	0.52 ± .03 %	optimal	1.48 ± .11 %
185429 ± 164	0.52 ± .04 %	79440 ± 221	1.48 ± .11 %
4193 ± 38	0.62 ± .03 %	1479 ± 28	1.48 ± .11 %
2043 ± 39	0.89 ± .03 %	433 ± 31	1.49 ± .09 %
1545 ± 35	1.17 ± .03 %	172 ± 7	1.49 ± .11 %
1328 ± 65	2.24 ± .05 %	122 ± 33	1.60 ± .12 %
1248 ± 62	4.89 ± .31 %	37 ± 1	1.76 ± .09 %
min-hop	8.26 ± .07 %	min-hop	4.06 ± .15 %

Table 4.1: This table gives the loss ratios for various update frequencies. The results are for a moderate workload on each topology.

each test, the algorithm was run both with and without admission control. The tests were run on both networks, with the smaller one requiring a higher interarrival rate to generate an equivalent workload. There were six different update levels tested. The table shows how the performance of the algorithm degraded as updates became more infrequent. The table compares performance with different update levels against the performance of the centralized implementation operating with perfect knowledge. The results using min-hop are also included to show how *Route_or_Block* performed better.

Generally, the results from these tests were as expected, but they also had some curious trends. The results in Table 4.1 are representative of the overall patterns in the data collected. As database error increased, the algorithm performed worse. This comes as no surprise. Interestingly enough, however, the relationship between error and performance is not linear. The systems can tolerate a certain amount of error without much performance degradation. This is more true of the sparse network, which is probably due to the fewer alternatives available for each decision. As expected, average hop counts for connections in the sparse graph were greater than those for the dense graph. This is obviously due to the lower connectivity.

The next step was to see if the previous trends continued to hold true for any kind of workload. The same series of experiments was performed on a range of workloads,

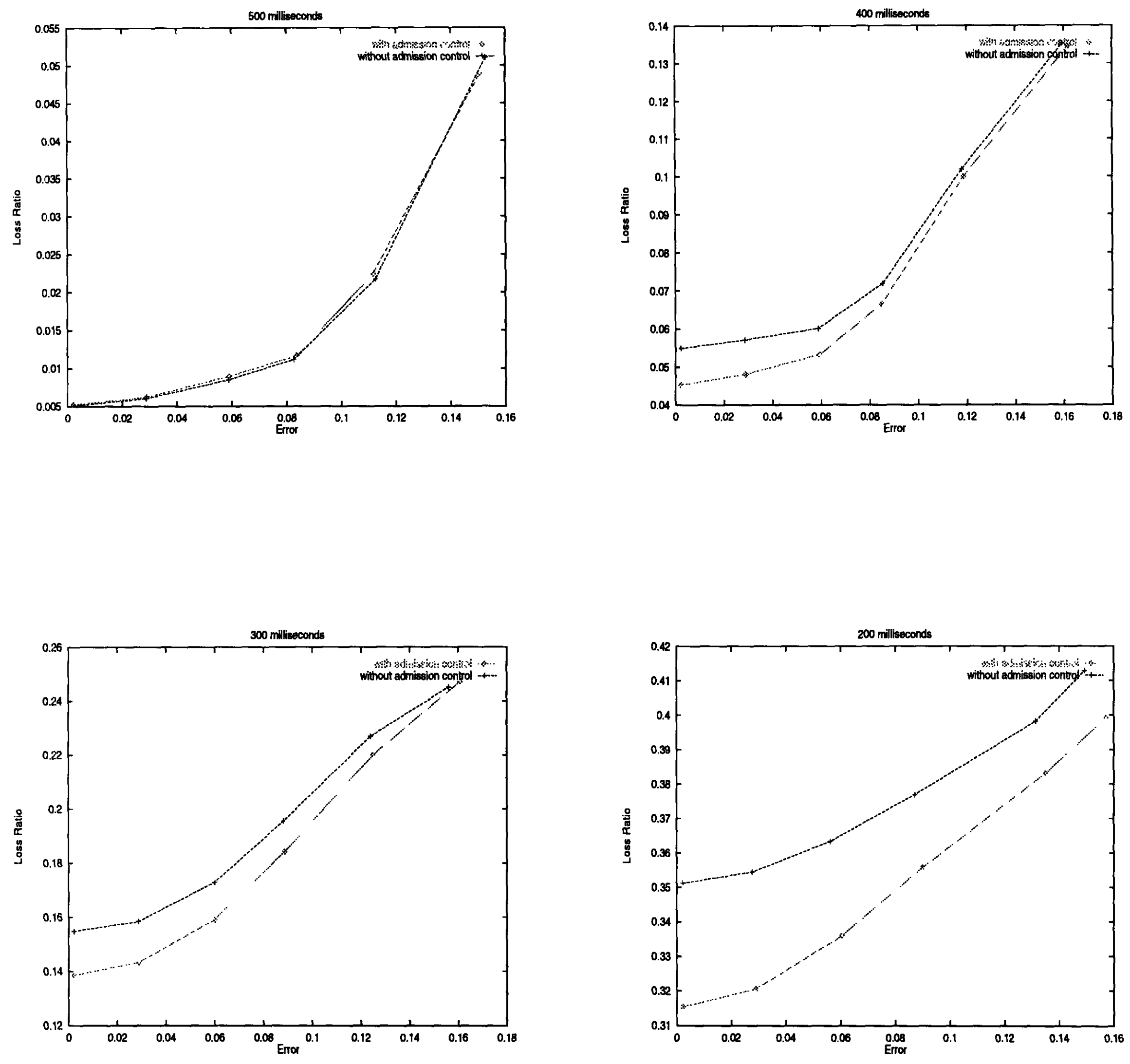

Figure 4-1: These graphs show the relationship between average database error and loss ratio. They show how performance degrades as the local databases become less accurate.

from very lightly loaded to heavily loaded. Four interarrival means were chosen. The results can be seen in Figure 4-1.

For the most part, the outcomes were similar. Unless the network is very highly stressed, the algorithm can withstand some error without undue degradation of performance. The algorithm only became very sensitive to error when so many connections were requested that every third request was dropped. At that point, the relationship was almost linear. This linearity stems from the fact that the system is saturated, and every increase in database error leads to a corresponding increase in loss ratio. In a realistic network, performance is rarely allowed to degrade that far.

The fact that under normal conditions the system was very robust is an encouraging result. Interestingly, the loss ratio increased swiftly as the workload grew. When the incoming load doubled, from an interarrival mean of 400 milliseconds to 200 milliseconds, the loss ratio became over six times as great. Clearly, the relationship is not strictly linear either.

4.2 Update Frequency

The previous results relate only part of the story. Now that the relationship between database error and loss ratio has been established, it is necessary to look at the number of updates required to keep the error within certain bounds. This is the main quantity of interest, since it is the updates themselves that tax the resources of the network. So the main questions now are: what frequency of updating maintains this comfortable level of error, and does this frequency depend on the level of traffic or on the network topology? These questions will be answered by looking at the total number of updates which where broadcast during the execution of each simulation.

Here again the results follow a pattern. This pattern is that the average database error does not begin to increase dramatically until the frequency of updating falls below a certain level. This level remains fairly consistent for all types of workloads. This can be clearly seen on the graph in Figure 4-2. According to these results, the relative frequency of updates necessary to maintain algorithmic stability can remain

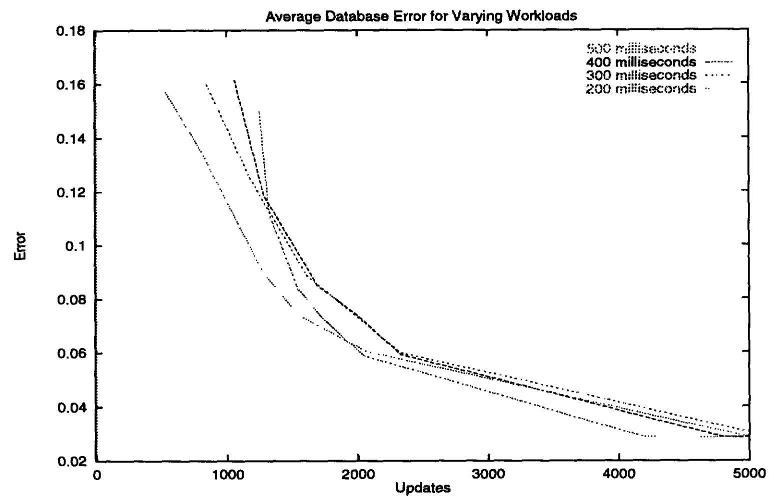

Figure 4-2: This graph shows the relationship between update frequency and database error on the dense topology. Each line represents a different level of traffic load.

effectively constant regardless of the amount of traffic flowing into the network. This avoids a cyclic effect where the number of updates would ebb and flow as the traffic pattern varies. Even if the amount of traffic entering the network isn't stable, the frequency of updating will be stable.

In fact, this level is even somewhat constant over both networks. The reason for this effect is difficult to explain. A possible reason might be that the value of information transmitted by each update increases as the workload does. In order to understand this relationship better, a series of tests were run with varying workloads yet equivalent update frequency. The results were then examined to see if each update corrected more error as the workload increased. The expectation was that as the load increased, each update would become more significant. However, the results did not bear this out. This is an interesting yet unexplained effect.

The previous results lead to the most interesting question. Namely, what is the overall effect of the relationships between error and performance, and frequency and error? The cumulative effect can be seen by examining the correlation between the number of updates and loss ratio.

The graphs in Figure 4-3 clearly show that the number of updates necessary to keep performance relatively high remains constant for any type of workload. The loss ratio will not increase to an unacceptable level until the frequency of updating has dropped significantly. No matter what kind of load is presented, or what topology the system consists of, a steady level of updating will result in near optimal performance. Likewise, having a very high number of updates does not improve operation by much, since overall performance is already close to optimal. This establishes the main pattern of the results, which should be close to the behavior of such a system in an actual implementation.

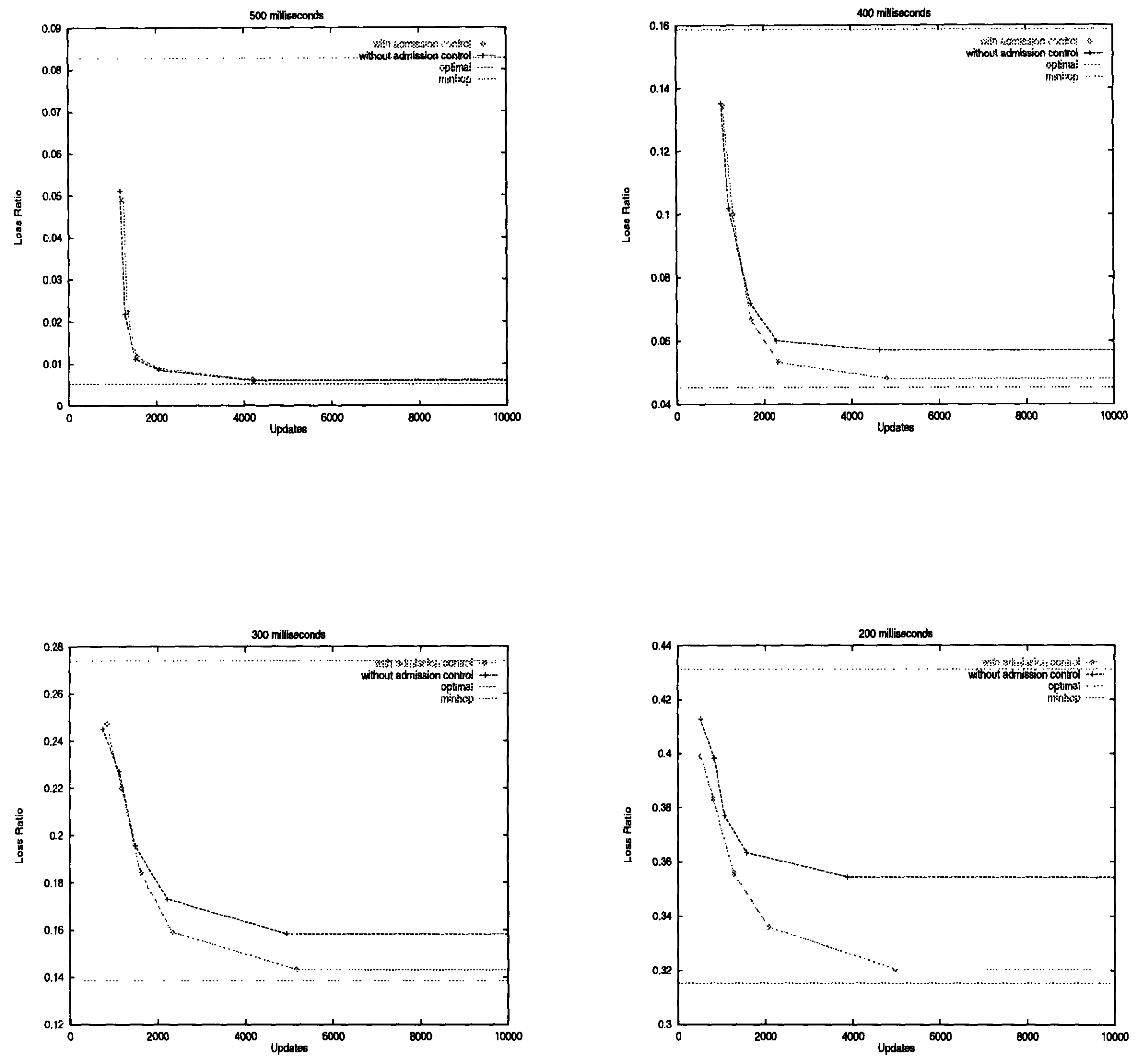

Figure 4-3: Each of these graphs shows the effect on performance as the level of updating decreases. The graphs compare these relationships over four different workloads on the dense topology.

Chapter 5

Local versus Global Updates

This chapter explores the question of whether local knowledge is more significant than global knowledge. That is to say, how important is it for each database to have accurate knowledge about distant edges in the network? Since most connections utilize paths that are not very long, the algorithm may be able to operate effectively with only limited global information. This suggests the possibility of using only local updates in the network.

These are updates which travel only 2 hops from the source node. A global update requires that the source node broadcast the message to all other nodes. For a local update, a node would only flood the area within a short distance. These limited updates would maintain accurate information about nearby edges while allowing a higher level of error for distant parts of the network. Correspondingly, this type of update utilizes fewer resources than conventional updates, which must broadcast throughout the entire network. The motivation is to trade potentially useless information for conservation of resources. Another advantage would be faster updates.

There are two sections in this chapter. The first covers the effect of local updates on the algorithm. The goal was to discern how well the algorithm performed with only limited global knowledge. The second section checks to see if this effectiveness is dependent on the network topology. In a sparse network, the paths of connections would tend to be somewhat longer. Also, the 'spread' (the number of nodes reached) of a local update would tend to be smaller.

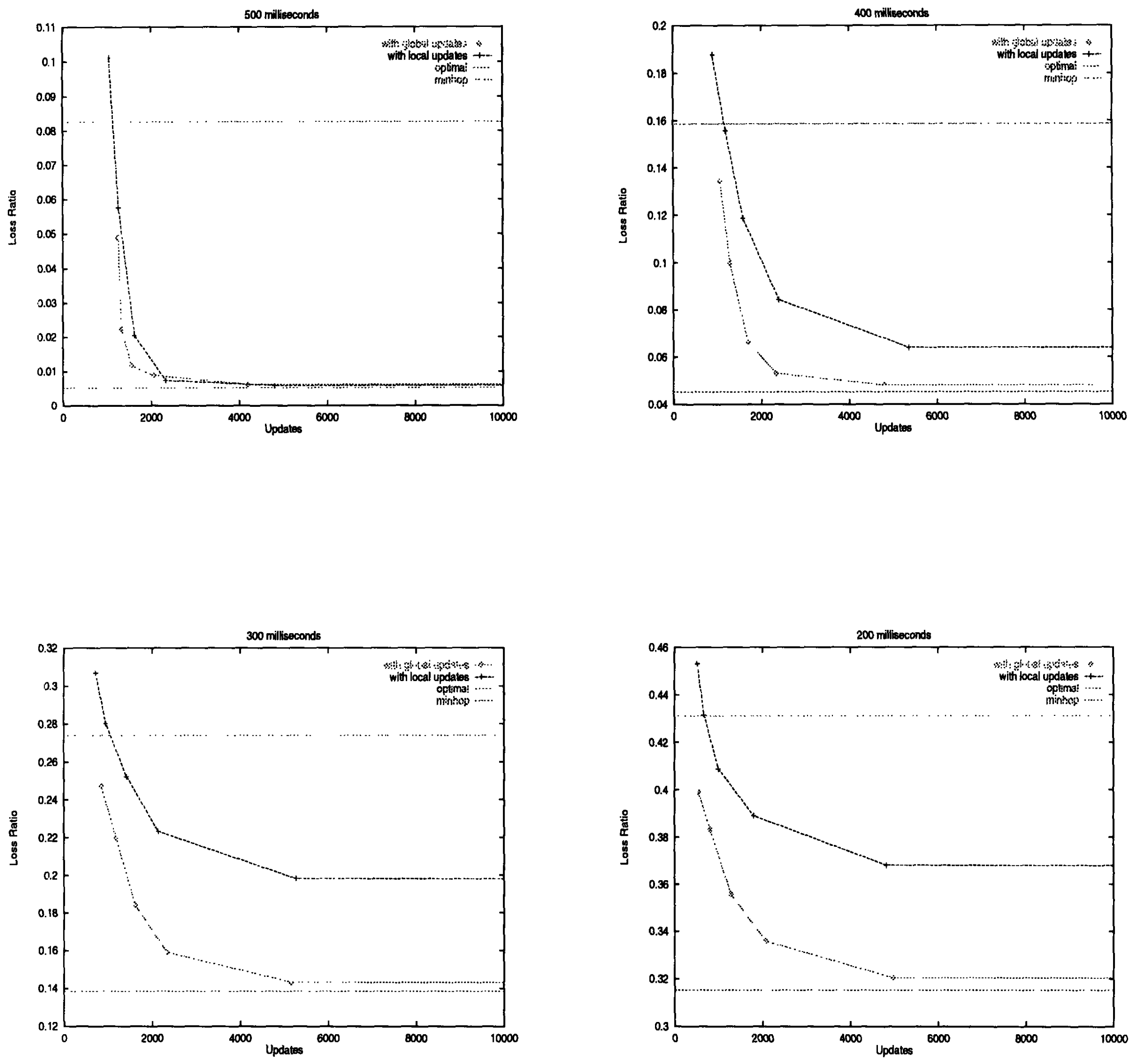

Figure 5-1: These charts contrast the performance of the algorithm using global updating against the algorithm using only local updates.

5.1 Sensitivity to Global Knowledge

The same series of experiments used in the previous set of experiments were run again on the dense network. On this occasion, however, only local updates were used instead of global broadcasts. Once again, there were two variables of interest. The interarrival mean of the requests ranged from 200 milliseconds to 500 milliseconds, and the update threshold varied from 1% to 50% of capacity. Since the same incoming traffic patterns were used, the results can be compared to see how the algorithm did with local updates.

Two main points emerged from the data. The most important is that the algorithm did not do as well with only local information. This can be seen from Figure 5-1, which contrasts the performance of the algorithm over several workloads and different frequencies of updating. Once the amount of traffic in the network passed a certain point, the algorithm began to perform less effectively using only local updates. This is similar to the results which Bar-Noy and Gopal found, where routing algorithm performance decreased if the nodes had only local information [5]. This effect becomes more pronounced as the network load increases.

Also, the update frequency became more significant. For example, at light loads with frequent updates, the two versions of updating produced almost the same results. However, as the number of updates grew less frequent, the performance of the algorithm with the local updates declined at a faster rate. The simulator reached a similar steady-state as it did with global updates. But this steady-state was at a lower level of performance than previously. So even when the frequency of updating was very high, the algorithm still was not able to approach optimal performance. This effect became more pronounced as the workload increased. At very light system utilization, the difference between the two methods was not very noticeable. But as the traffic increased, the simulation using local updates had higher and higher loss ratios.

The other trend concerned the stability of the algorithm. Unlike test runs with global updates, the comfortable margin of error did not remain the same as the traffic increased. A larger number of updates was required to maintain database error within certain limits. This can be seen from Table 5.1, compared against the graph of error in Figure 4-2. For each workload, the table gives the average database error for the highest and lowest frequencies of updating. Using global updates, the upper and lower bounds for error remained constant independent of workload. With local updates, the bounds increased as the load increased.

The increase in error due to higher workload relates to the disparity between performance at low and high workloads. If the overall system error was not too great, the local update version did almost as well as the system with global updates. But

λ	*lower bound*	*upper bound*
	dense graph	
500	25.09 ± .04 %	33.39 ± .31 %
400	29.61 ± .03 %	35.34 ± .19 %
300	31.46 ± .02 %	37.19 ± .26 %
200	32.13 ± .01 %	38.99 ± .22 %
	sparse graph	
2000	28.75 ± .13 %	30.29 ± .34 %
1500	35.79 ± .07 %	40.01 ± .59 %
1000	45.96 ± .06 %	46.93 ± .78 %
500	52.58 ± .02 %	53.24 ± .38 %

Table 5.1: This table shows the average database error for the most frequent (the lower bound) and the least frequent (the upper bound) levels of updating. These ranges are examined for each traffic load.

as the utilization increased, so did overall error, which caused the local updates to become less and less effective. The update frequency necessary to approach steady-state also tended to increase. As the data shows, using only local updates makes the system highly sensitive to the traffic load on the links.

5.2 On a Sparsely Connected Network

To understand whether the use of local updating would be effected by network topology, a series of tests were run on the sparse graph. These tests were identical to the previous simulations run on this network. The aim was to see, by comparing these results to the ones for global updates on a sparse network, if there was any difference attributable purely to the change in topology.

As with the experiments run on the dense network, the simulations using local updating had higher loss ratios. At low loading and low error, the difference was not very pronounced. Under these conditions, both systems approached optimal performance. But as database error increased, the loss ratio for the algorithm using local updates quickly rose. The error for the local method increased both with load and update threshold, so both of these factors decreased the performance. As pointed

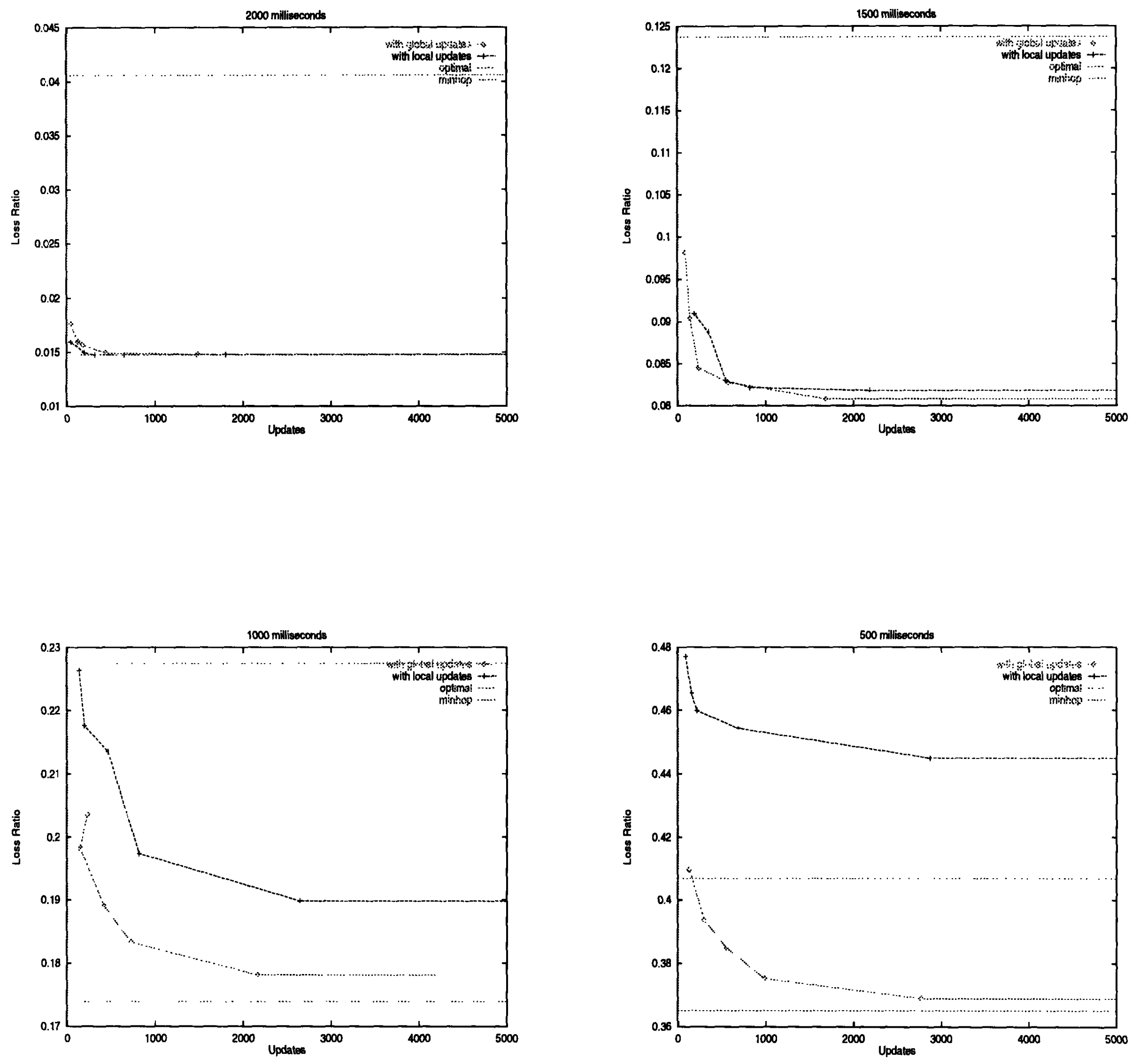

Figure 5-2: These graphs compare the performance of the algorithm using local updating against global updating on the sparse topology.

out before, when using global updates, database error does not increase with traffic. So the algorithm became as or more sensitive to both error and utilization, as it had on a dense one.

However, the surprising result was that the type of network topology did not seem to have much effect. While using local updates is not optimal, the loss ratio will not increase simply because the network has fewer available paths. In fact, for both methods of updating database information, the impact of error was not as great on the sparse network. Comparing the results of all simulations run on both networks, the simulations on the dense network were more effected by inaccurate databases. This agrees with the theoretical results produced, which showed that the performance is dependent on the network size. In a dense network topology, the number of available paths is greater for each new request, which would increase the significance of accurate local databases.

Chapter 6

Discussion

The experiments and theories given in the previous chapters detail the effects on the system from incomplete knowledge about link states. These results illustrate how decision making at access points into the network is dependent on this information. This chapter attempts to extract the central meaning from that work, and point to some new directions for future research.

The most interesting result is that the system is relatively robust. This point was made both in the analytical proofs and in the empirical experiments. In Chapter 2, the two theorems proven showed that *Route_or_Block* not only operates correctly in a distributed implementation, but that its performance is comparable to the performance of the centralized version. Both are $O(\log nT)$ competitive compared to the off-line algorithm. As mentioned previously, this is optimal for any on-line routing and call admission algorithm [1].

The experimental results from Chapter 4 showed how the algorithm is effected by error when making decisions locally. The results clearly showed that while a centralized, omniscient implementation has optimal performance (the minimum loss ratio), the distributed implementation can approach this level as well. Also, the frequency of updating necessary for near-optimal performance can be low without adversely effecting the system greatly. (Approximately an order of magnitude less frequent than connection arrivals.) These results held for various network topologies and workloads.

Therefore, *Route_or_Block* is a suitable routing and admission control algorithm for a high speed wide area network. In an implementation of an ATM internet backbone, this algorithm would effectively outperform shortest path routing strategies by reducing the overall network congestion and improving access to the network. There would need to be some sort of topology update algorithm to maintain link state information, but this could be very simple. The amount of control traffic necessary wouldn't be a heavy load on the system. The *Route_or_Block* algorithm is also very stable, since it does not seem to be subject to any oscillatory behavior. None of the simulations showed any tendency for the algorithm to cycle between overloading various paths. Finally, this implementation would be very scalable. This is because the competitive ratio is only logarithmically dependent on the network size, and also because the algorithm is computationally efficient.

However, this thesis did not study the possible implications of using this algorithm on a network consisting of mixed transmission media. The Internet in its current incarnation exists consists of combinations of almost all types of channels, from fiber optic trunk lines to Ethernet local area networks to satellite links. The ATM protocol is designed to operate effectively over the varying bandwidth capacities and latencies of these different channels. Whether or not *Route_or_Block* is capable of managing the resources effectively in such an environment is currently an open question.

The second main conclusion of this work is that routing decisions depend almost as heavily on global knowledge as they do on local. This was shown by the failure of the *Route_or_Block* algorithm to perform well with only a local update strategy. There was some benefit to using local updates at very light workloads. However, this may not be that significant for an actual application. Cells can be switched and transmitted very quickly in fiber optic networks, so the overhead from using global broadcast isn't very great. The resource savings from using local updates might not be very important.

Conceivably, there might be some advantage from using a mixture of various update strategies, as opposed to the all or nothing approach examined here. This could also be impacted by the fact that traffic probably isn't uniformly distributed over the network. Traffic would tend to be concentrated locally, but to what extent is difficult to say. Examining this question of possible tradeoffs between local and global knowledge could use some further study.

Obviously, there exists many starting points to explore new areas from the research in this paper. Hopefully, there will be some attention paid to the insights gained from this research. The interaction of routing, call admission and topology update is an important problem in the implementation of the next generation of high speed wide area networks. This paper showed that *Route_or_Block* is a distributed, scalable algorithm which has some promise as a solution to this problem.

Bibliography

[1] Baruch Awerbuch, Yossi Azar, and Serge Plotkin. Throughput-competitive on-line routing. In *Proc. FOCS*. 1993.

[2] Baruch Awerbuch, Israel Cidon, Inder Gopal, Marc Kaplan, and Shay Kutten. Distributed control for PARIS. In *Proc. 9th ACM Symposium on Principles of Distributed Computing*. 1990.

[3] Baruch Awerbuch, Rainer Gawlick, Tom Leighton, and Yuval Rabani. On-line admission control and circuit routing for high performance computing and communication. Unpublished manuscript, 1994.

[4] Saewoong Bahk and Magda El Zarki. Dynamic multipath routing and how it compares with other dynamic routing algorithms for high speed wide area networks. In *Proc. SIGCOMM*. 1992.

[5] Amotz Bar-Noy and Madan Gopal. Topology distribution cost vs. efficient routing in large networks. In *Proc. SIGCOMM*. 1990.

[6] David Clark, Bruce Davie, et al. The AURORA gigabit testbed. In *Proc. INFOCOM*. 1992.

[7] Ajei Gopal, Inder Gopal, and Shay Kutten. Broadcast in fast networks. In *Proc. INFOCOM*. 1990.

[8] Pierre Humblet, Stuart Soloway, and Bradford Steinka. Algorithms for data communication networks. Technical report, Codex Corporation, 1986.

[9] Bruce Kim and Peter Wang. ATM networks: Goals and challenges. *Communications of the ACM*, 38(2), February 1995.

[10] Ian Leslie and David Tennenhouse. A testbed for wide area ATM research. In *Proc. SIGCOMM*. 1989.

[11] Frank Lin and James Yee. A real-time distributed routing and admission control algorithm for ATM networks. In *Proc. INFOCOM*. 1993.

[12] Gopal Meempat and Malur Sundareshan. A hierarchical scheme for combined access control and routing of circuit-switched traffic in ISDN environments. In *Proc. INFOCOM*. 1991.

[13] Ronald Vetter. ATM concepts, achitectures, and protocols. *Communications of the ACM*, 38(2), February 1995.