--   --

Chapter 4: Evaluation of fault recovery

4.1. Introduction

The problem of network fault recovery has been extensively studied. Chapter 1 provides a survey of existing work exploring various aspects of fault recovery in the context of telecommunication networks and computer data networks. The field is relatively mature and a good understanding of the issues involved in recovering from fault in these networks exists. However, as we saw in Chapter 3, the assumptions for realtime networks are considerably different. The situation is closest to that found in the cross-connect layer of telecommunication networks, but the solutions used there cannot be applied directly to the new environment without considerable thought and experimentation. In Chapter 3, we looked at the issues involved in redesigning the fault recovery mechanisms to work with realtime networks. We presented some initial design choices, which narrowed the solution space, and then proposed some orthogonal factors that would allow us to explore the remainder systematically. In this chapter, we present the results of the simulation experiments that we performed. We first describe, in Section 4.2, the simulation tools, and then present and justify the experimental methodology. This includes identifying metrics of comparison that reflect the objectives of fault recovery specified in Chapter 3. We also identify the internal and external factors that are expected to effect the performance of the schemes. In Section 4.3, we present a series of experiments and results, by which we explore the chosen solution space for algorithms to reroute realtime connections after a network failure. Our conclusions from these experiments are summarized in Section 4.4.

4.2. Simulation design

The recovery schemes were designed with as few assumptions about the underlying realtime schemes as possible. However, in order to evaluate the sensitivity of the schemes to the external factors (such as the topology, the load, the traffic mix, and the number of faults) and to pick the best combination of approaches for the internal factors of the scheme, we need to build a simulation model which specifies all the details of the underlying realtime network. Fortunately, the author had access to a simulation package for realtime communication on packet switching networks written by Hui Zhang. This was extended to include signaling and admission control using the Dynamic Channel Management protocol by Colin Parris. This simulator provides a detailed packet level simulation of the extended realtime channel scheme described in Section 2.5. The schemes for fault recovery described in the previous chapter were implemented on top of this platform by the author.

In this section we describe the framework of the set of experiments performed using this simulator. We first present an overview of the simulator. Then we consider the details of the realtime network being simulated, and briefly describe the scheduling mechanism used for packet forwarding and the admission control tests used during channel establishment, as implemented in this particular simulator. In the context of the scheduling discipline described, we present a metric of load that eases the task of comparing the performance of different rerouting schemes. We define metrics of performance that reflect the objectives of fault recovery presented in Section 3.2. Then, we describe the methodology of the experiments by identifying the internal and external factors which affect our chosen metrics. The external factors such as topology and load are elements which are not under the control of the scheme; thus, we study the sensitivity of the schemes to these factors. The internal factors are the variables within the scheme that we discussed in the previous chapter. The simulation experiments are intended to measure the effect of each factor on the overall success, as well as to evaluate the choices that can be made within each factor, in order to maximize the reroute success. This forms the body of our simulation experiments, where we systematically try the various combinations of approaches to each factor, and observe the effect on our metrics of performance.

4.2.1. The simulator

The simulator is based on the DCM simulator built by Colin Parris and Hui Zhang (51). It allows the user to specify a network topology, by defining the set of vertices of a graph (which correspond to the routers and hosts in the network) and the set of edges which join the vertices (which correspond to point-to-point links in the network). The edges have properties such as link speed, and scheduling discipline, which are specified by the user. The user also specifies a set of initial network events, which cause other events to be triggered in the simulator. The main type of event specified by the user of the DCM simulator is the channel establishment event. Each channel establishment event is characterized by the traffic and performance parameters of the channel to be established. The traffic parameters supported by this simulator are $Xmin, Xave, I$, and S_{max}. The simulator only supports deterministic delay bounds; thus, the only performance parameter is D_{max}.

The channel establishment event causes the simulator to trigger a round-trip channel establishment. When the establishment is complete, packet transmission events are triggered from the source node at intervals defined by the traffic parameters. This simulates data transfer along the channel.

The DCM simulator also allows its user to specify alternate establishment events, in which an existing channel is subjected to parameter or route modification. The alternate establishment event requires the user to specify the new traffic and performance parameters; a new route may be optionally specified.

Thus, the simulation is started with a set of events from an event file, each of which is associated with a time and other necessary parameters. As the simulation progresses, these events are processed in chronological order. The processing of an event such as a request for establishment in turn causes several events to be created, such as a message being sent from the source of the request to its neighbor, or the generation of data packets after the successful establishment of the channel. All events are maintained in a calendar queue (13), and, as simulation time elapses, the next event is removed from the

-- --

queue and processed.

The events simulated in the simulator include:

- Arrival, queueing, and scheduling of packets in nodes
- Transmission and propagation of packets on the links
- Arrival of requests from clients leading to channel administration messages being sent to neighbors
- Arrival of channel administration messages leading to admission control tests or tear down calculations being performed
- Arrival of route update messages leading to modification of the routing tables

The author modified the above simulator to permit the specification of fault events. When a fault event occurs, the simulator automatically generates the necessary alternate establishment events necessary to reroute the affected channels. The route, the parameter specifications, and the time of the event are generated according to the rerouting scheme being simulated.

A rerouting scheme is the combination of the approaches to each of the internal factors described in the previous chapter, i.e., one scheme may be Immediate timing, Global locus of reroute, No retries, and Local prediction. The input file allows each of these internal factors to be set to a particular approach. The simulator models the chosen scheme during the simulation, selecting routes, timing the establishment attempts and retries, and performing state prediction automatically. Thus, the simulator can model any of the schemes in the solution space depicted in Figure 2.5. All the approaches to the internal factors described in the previous chapter are implemented. The reasonable schemes in the region of the solution space described in the previous chapter were tried out in simulation.

The simulator also models the route update protocol, which modifies the local databases on which the nodes in the network make routing decisions, using a reliable broadcast method as described in Section 2.5.3. This allows us to study the effects of incomplete or out-of-date information, and the performance of algorithms to counteract these effects, in our experiments.

The simulator does *not* model the computational times for running the algorithms on the nodes. This time is assumed to be zero in the simulation model. This can be justified by measurements of the time to run the various pieces of code, as implemented in the simulator and run on a DECstation 5000/240 (53). The time to perform the simple RCSP admission control tests (described in Section 4.2.3) in the nodes on the forward and reverse pass of the establishment process (measured value = 3.3 μs) is negligible compared to the link propagation delays (10 ms). Since one execution of the test is performed for each link crossed, the inclusion of this computation time in the simulation would make a very small difference to the results. The time to run the routing algorithm on the "mesh" topology (Figure 4.3) is around 1 ms, which is small compared to the round-trip time to perform the establishment (average $\approx$ 100 ms). Again, since routing is performed once per round-trip establishment attempt, the inclusion of this computational latency in the simulation model would make no more than a few percent difference in the results.

4.2.2. The scheduling discipline

One of the fundamental processes being simulated is packet scheduling. As control or data packets arrive at a node in the simulator, they are queued for service, and served in the order specified by the scheduling discipline at the node. The link (together with the scheduling mechanism) can be viewed as a queueing server; there is one such server for each link on each node of the network. The scheduling discipline used for packet forwarding in our experiments is Rate Control Static Priority (RCSP) scheduling (71). We chose to use RCSP in our simulations for a number of reasons. RCSP separates the issues of rate control and delay control by conceptually dividing the RCSP server into two modules, as described below. This separation makes it possible to independently allocate delay and bandwidth, allowing simple and fast admission control tests. This simplicity allows us to analyze the effect of load on an RCSP network, making it easier to design metrics of load and performance, which we use to present our results. Finally, the simpler model facilitates our understanding of simulation results and eases the task of recognizing unreasonable results. This was especially useful in the earlier stages of our investigation, when we were debugging the simulator.

The rate control module enforces the traffic specification constraints on a per channel basis; if streams become more bursty (such as might happen as a result of passing through a number of servers), they are smoothed out by the rate controller, so that they will obey the traffic constraints again. This also serves to police the sources at the inputs to the network. The static priority module assigns each channel to one of several priority levels, each of which has an associated delay bound. Each priority level conceptually has a separate queue, and packets are served from the highest queue with a non-zero queue length. The admission control tests are responsible for keeping the admitted load for each level below the point where packets would violate the associated delay bounds. Figure 4.1 shows a conceptual picture of the operation of the RCSP scheduler, reproduced from (71).

4.2.3. The admission control tests

The admission control tests implemented in the simulator enforce a condition for call admission that is sufficient, but not necessary, for the satisfaction of performance guarantees during packet forwarding. In other words, the bounds are not the tightest ones that can be obtained. The calculations only take into account X_{min}; thus, the bandwidth reservation is peak rate. However, the tests require minimal computation during establishment. They are based on the following theorem, which is reproduced from (71).

Theorem 4.1: Let $d_1, d_2, \cdots, d_n$ $(d_1 < d_2 < \cdots < d_n)$ *be the delay bounds associated with each of the n priority levels, respectively, in an RCSP server. Assume that the* j^{th} *connection among the* i_k *connections traversing the server at priority level* k *has the traffic*

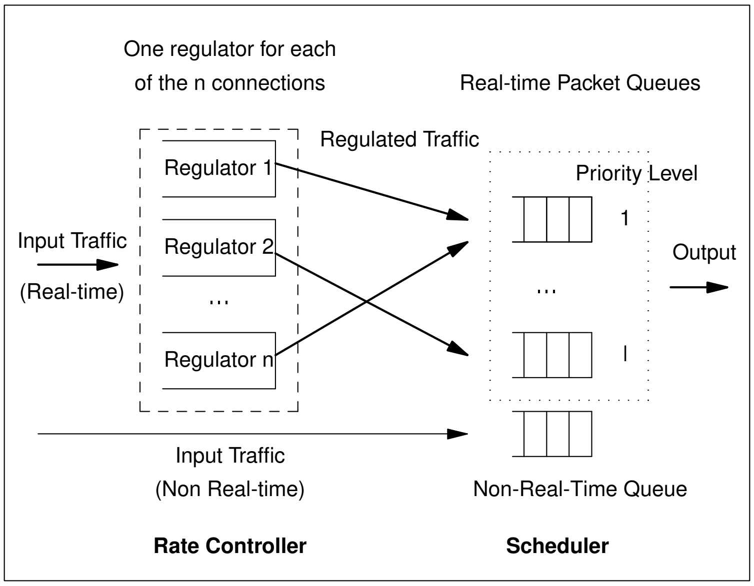

Figure 4.1: Rate Control Static Priority scheduler

specification ($X_{min}^{k,j}, X_{ave}^{k,j}, I^{k,j}, S^{k,j}$). Also, assume that the link speed is l, and the size of the largest packet that can be transmitted onto the link is S_{max}. If

$$\sum_{k=1}^{m} \sum_{j=1}^{i_k} \left\lceil \frac{d_m}{X_{min}^{k,j}} \right\rceil S^{k,j} + S_{max} \le d_m l \tag{4.1}$$

the waiting time of an eligible packet at level m is bounded by d_m.

Theorem 4.1 gives the admission control condition that needs to be tested. For each priority level m, a state variable B_m is maintained and initialized to S_{max}. When an establishment request for a new connection with traffic specification (X_{min}, X_{ave}, I, S) comes in, the following tests are performed for $m = 1$ to n.

$$B_m + \left\lceil \frac{d_m}{X_{min}} \right\rceil S \le d_m l \tag{4.2}$$

The new connection can be placed into priority level k without jeopardizing the performance guarantees of other existing connections if and only if Equation 4.2 holds for $m \ge k$. If the condition is satisfied and the decision to put the connection into level k is made, then B_m is updated for $m \geq k$ as follows:

$$B_m = B_m + \left\lceil \frac{d_m}{X_{min}} \right\rceil S \tag{4.3}$$

The intuitive interpretation of these tests is that Equation 4.1 computes a bound on the maximum busy period of the queue at priority level m. This is defined as the maximum contiguous period of time during which the queue at level m can contain one or more packets. A busy period starts when a packet arrives at an empty queue, and ends when the last packet in the queue is transmitted. Since the maximum delay any packet at this level can suffer is bounded by the maximum period, this serves as a sufficient (but not tight) upper bound on packet delays for packets on this level. The last term on the left hand side of Equation 4.1, S_{max}, accounts for the possibility that a best-effort packet was already in service when the busy period started, since the service is non-preemptive. Since the tests ensure that the new channel will always be rejected if the admission of the new channel causes the busy period bounds to exceed the delay bounds for any level, we can be sure that no packet in a channel belonging to level m will suffer a delay bound greater than d_m.

The above test takes care of the local delay bound for the channel. To ensure that sufficient buffers exist, a buffer test is also performed, and the required memory is reserved for the use of the channel. The details of the buffer test are not mentioned here.

4.2.4. The load index

One problem facing any investigation that attempts to compare schemes involving realtime channels is the one of choosing an index that characterizes the load imposed on a network by one or more channels. Finding a single number that captures the load is hard because it depends on at least the following components of the load: the number of realtime channels, the number of hops each channel traverses, the traffic specifications of each channel (which has peak and average rate components), and the delay requirements of the channels. A general solution to this problem is quite complex, and some simplification is usually chosen. For example, the sum over all existing channels of the product of the average rate (in bits per second) of the channel and the number of links over which the channel passes could be used as a load index:

$$\sum_{j=1}^{N} \frac{S^j I^j H^j}{X_{ave}^j} \tag{4.4}$$

where N is the number of channels existing in the network, S^j, X_{ave}^j, and I^j are from the traffic description of channel j, and H^j is the number of links channel j traverses. This would capture the average rate and the number of links affected. However, this load metric would completely ignore the delay requirement of the channels.

Fewer low delay channels can be supported by a realtime network. Thus, if a low delay channel is accepted, more of the resources are reserved, and fewer channels can be subsequently accepted. The reservation due to delay bounds is harder to quantify than the reservations for buffer or bandwidth, but the admission control tests of Equation 4.1 express it algorithmically for the RCSP scheduling discipline. The establishment of a channel leads to a decreased capacity of the network to accept further realtime connections, and this decrease in capacity is greater if the delay bound required is lower. A good load metric should be able to quantitatively capture this effect. The exact mathematical properties of the reduction in realtime capacity depend on the admission control tests being used, which in turn depend on the packet service discipline in the network elements. The general problem is, thus, considerably harder, and we restrict ourselves to the specific case of Rate Control Static Priority (RCSP), with the simple admission control tests described above.

The static priority module of an RCSP server consists of n priority queues, each of which has an associated delay bound. For each level, the admission control mechanism maintains a variable, B_m (Eq. 4.3), which bounds the busy period for the corresponding level; the test consists of recomputing B_m for all levels $m \geq k$ when a new channel is introduced at level k, and ensuring that the new values are less than the corresponding delay bounds (d_m). Thus, adding a new channel at level k increases B_m for each level $m \geq k$ (Eq. 4.3). The busy period bound B_m is, therefore, a direct measure of the load on a particular server at level m. Since the placement of a channel at a higher priority (lower numbered) queue effects all lower priority (higher numbered) levels, the sum of B_m over all levels, if used as a index of load, has the desirable property that the placement of a channel at a higher priority level has a larger affect on it. Since $B_m = S_{max}$ in the absence of any channels, we need to subtract S_{max} from B_m to make the load index zero at a server without any established channels. We can normalize the index by dividing by the link speed, l, to make it it possible to compare the loads on links with different speeds. Finally, if we want to take into account the effect of load on the entire network, we can add the index at a server over all servers in the network. There is one RCSP server for each directed edge in the network graph (one in each direction for every link). The final index of load is defined as follows:

$$load(t) = \sum_{e \in \mathbf{E}} \sum_{k=1}^{n} \frac{B_{k,e}(t) - S_{max}}{l_e}, \tag{4.5}$$

where $load(t)$ indicates the network load at a particular time t, and changes as channels are set up or torn down, $\mathbf{E}$ is the set of directed edges in the network, n is the number of levels in the RCSP priority queue module, $B_{k,e}(t)$ is the value at time t of the maximum busy period as defined in Equation 4.3, for level k at the server for the directed edge e, and l_e is the link speed for the directed edge e. Load is measured in time units.

This index of load has many desirable properties. It is

- monotonically decreasing as a function of the delay requirements of the channels if all other parameters are kept constant
- linearly increasing in the peak rate requirement of the channels (since the reservations are peak rate)

- linearly increasing with the number of links affected
- additive to a first approximation over sets of channels

The additive property only holds if the channels in the set do not interfere with each other, as is the case when, for instance, a channel takes a different route or a different priority level because some other channel occupies a given set of resources. These properties are shown in Figure 4.2.

Apart from the intended use of this load index to design metrics to compare recovery schemes, we can think of several other applications of the index. For instance, since the index captures the effect of a channel or a set of channels on the network, it could be used for pricing. Such a pricing scheme, where the price of service is directly related to the effect on the network, has several advantages. For example, it would encourage users to use the network efficiently, since minimizing the cost to the user has been linked to minimizing the effect on the network. In addition, since the metric captures the resource state of an RCSP server in a single number, it may be used to guide the route selection process. This would reduce the amount of information to be carried for each link by the route update protocol, and also reduce the computation to be performed for route selection, which may be valuable for extremely large networks. We will not explore these possibilities any further in this dissertation.

4.2.5. Metrics of comparison

In order to compare the performances of different schemes, we have to be able to quantitatively define what we mean by good performance. There are several orthogonal criteria that must be considered. The amount of realtime load successfully recovered is a major concern, since it has a significant impact, to both the network provider and the clients. The time to perform the recovery is another criterion of importance. Finally, the effect on network resources is of concern to the network provider, since the recovery may come at the cost of significantly lower remaining network capacity after the

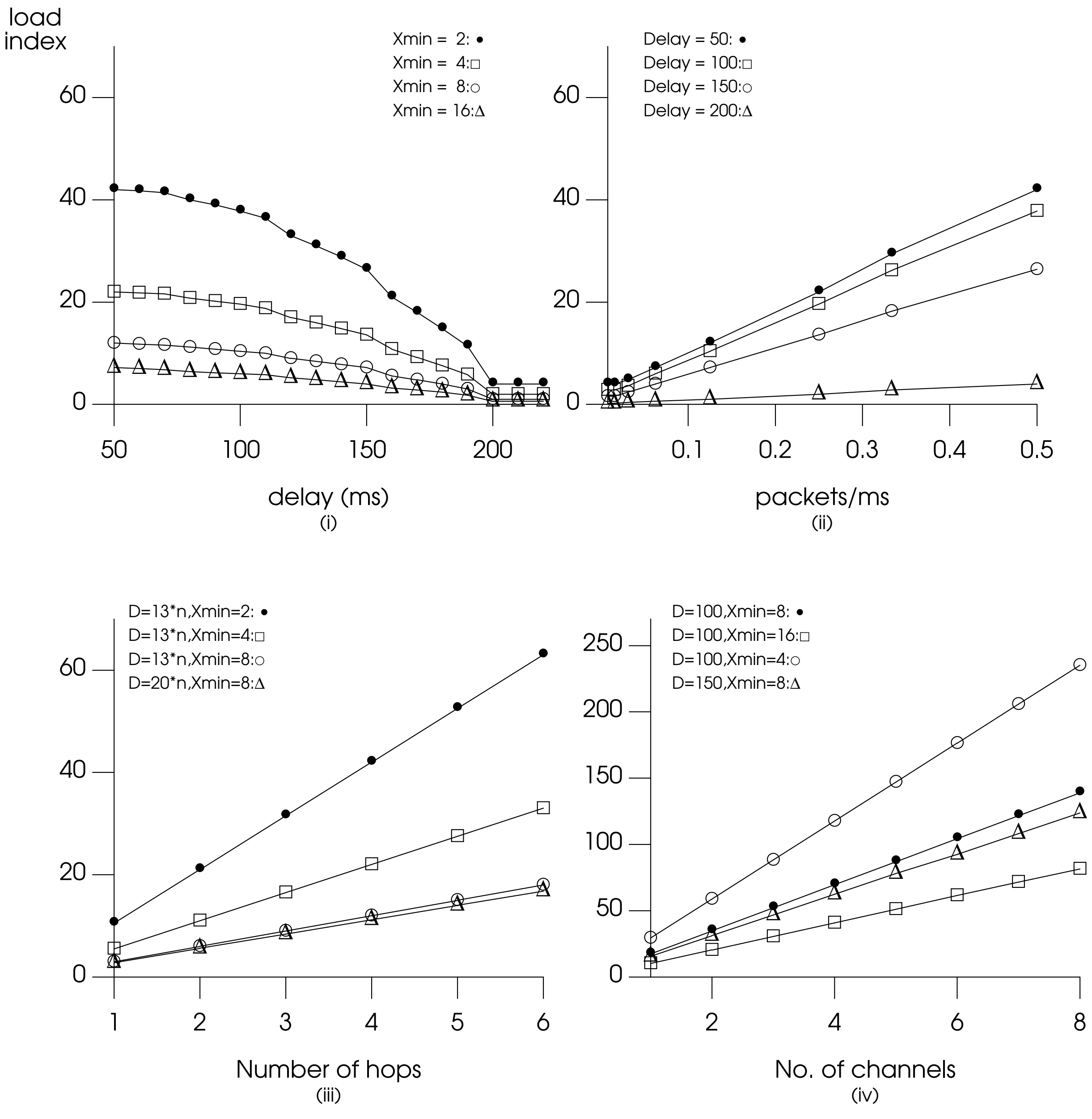

Figure 4.2: Properties of the load index

recovery process has completed, if the resources are inefficiently allocated.

Fraction of realtime load rerouted

We would like to compare the amount of realtime load successfully rerouted by different rerouting schemes, but this is hard to do unless the term "amount of realtime load" is well defined. The problem here is that the sets of channels are heterogeneous, with different traffic specifications and performance requirements. Hence, it is difficult to say if the amount of realtime load represented by one set is larger or smaller than another, unless we reduce all these factors to one number. The load index defined in Section 4.2.4 allows us to do that.

We first define *load*(c), the amount of realtime load contributed by a single channel c, for a given network topology and assuming a well defined deterministic routing algorithm, as the load index as defined in Equation 4.5, measured when the channel is set up in the network with no other channels established in it. Of course, this is only well defined if the routing algorithm is deterministic, i.e., always computes the same route given the same input. The Bellman-Ford algorithm described in Section 2.5.4 and used in our simulations meets this requirement.

load(c) is the minimal increase in network load that the establishment of channel c can cause on this network, and is indicative of the intrinsic resource requirements of the channel. The difference in the load index measured in a non-empty network might be larger than this, due to the channel taking a longer path or being assigned to higher priority levels if the lower levels are occupied. However, under the assumption that the routing algorithm always picks the most efficient path and the relaxation strategy always moves the channel to the lowest priority levels available, it can never be smaller. Thus, *load*(c) is a measure of the lowest possible amount of resources required by the channel c.

We define the amount of realtime load of a set of channels as the sum of the loads of the individual channels. Thus, the amount of realtime load corresponding to a set of channels **C** is:

$$load(\mathbf{C}) = \sum_{c \in \mathbf{C}} load(c) \tag{4.6}$$

Note that the load index in the network, even with a fixed set of channels **C**, will depend on the order and timing of the route computations and the establishments. On the other hand, *load*(**C**) is uniquely defined as long as the routing algorithms and establishment protocols are fixed. Its value is independent of the ordering of the channels. Under the assumptions that the routing algorithm and relaxation mechanism always pick the shortest paths and lowest priority levels possible, *load*(**C**) is less than or equal to *load*(*t*) after establishing the set of channels **C** in the network.[1]

Finally, we can define the metric to measure the fraction of realtime load rerouted following the occurrence of failure. The Success Ratio (SR) is defined as the fraction of affected realtime load that is successfully rerouted, expressed as a percent:

$$SR = \frac{load(\mathbf{R})}{load(\mathbf{A})} * 100 \tag{4.7}$$

where **A** is the set of affected channels, and **R** is the set of successfully rerouted channels.

Speed of recovery

The speed of recovery can be measured in terms of the average time to restore service to a channel in the event of a failure. This is measured over the set of successfully rerouted channels. The maximum time to recover any channel is also of interest, since this is a measure of the time for the network to stabilize, as well as an indication of the worst case recovery time. Both these numbers can be easily obtained from our simulations.

Efficient use of network resources

We can expect the rerouted channel to use more resources than the original channel, since the shortest path is blocked by the failed link. In general, the more highly loaded the network, the longer the alternate routes found will be, leading to more and more inefficient use of resources. However, the rerouting scheme may itself lead to longer reroute paths, and hence, to inefficient resource use. For example, the Local rerouting approach will always tend to use longer paths than the Global approach, and the

[1] We have overloaded the term *load*(X), but there should be no cause for confusion, since the type of the argument indicates which definition is to be used.

penalties of this must be assessed. We would like to do this in a way more general than comparing path lengths. The load index defined previously allows us to do that by defining the excess resources (ER) consumed by a set of reroutes performed on a network in the event of a failure as the difference between the actual resources used in the network to support the successfully rerouted channels using the set of paths chosen by the rerouting scheme and the minimum resources required to support the set of successfully rerouted channels **R**, expressed as a percent of the minimum resources required:

$$ER = \frac{actual - minimum}{minimum} * 100 \quad (4.8)$$

The resources actually used can be experimentally obtained (in simulation) by measuring the network load after the recovery process terminates, then tearing down the set of channels **R** and measuring the load again. The difference in the two loads corresponds to the resources actually used. The minimum resources required is approximated by *load*(**R**). Thus, we have:

$$ER = \frac{load(t_R) - load(t_{TD}) - load(\mathbf{R})}{load(\mathbf{R})} * 100 \quad (4.9)$$

where $load(t_R)$ is the load at the time the last successful reroute completes, $load(t_{TD})$ is the load after tearing down the set of successfully rerouted channels R, and $load(R)$ is the load as defined in Equation 4.6 for the set of successfully rerouted channels **R**.

4.2.6. Factors affecting performance

Now that we have defined what we mean by the performance of a recovery scheme, we can move on to identifying the set of factors that will affect the performance. The experiments will then consist of changing the factors and observing the effects on the performance indices.

The factors affecting performance can be broken into two classes. Those in one class, the external factors, are independent of the schemes, and we need to make sure that the schemes work well in the face of variations of these factors. This class of factors contains the network topology, the operating load on the network, the nature of the applications, and so on.

The second class consists of the internal factors of the recovery schemes. In the previous chapter, we have identified a set of orthogonal factors, such as routing constraints, timing, retries, and state prediction, that are expected to affect the performance of recovery in different ways.

4.2.6.1. External factors

The external factors that are considered during our simulation experiments are the network load, the failure mode, the network topology, and the traffic mix on the network. We will measure the performance of our schemes and how it changes when we vary the external factors. While, in general, a network will be designed keeping these factors in mind, so that in the common case the network is operating with desirable characteristics such as low blocking probability for new channel requests, or high percentage of recovery in the event of a failure, we should try to design schemes that degrade gracefully when the operating environment becomes harsher. Thus, one interesting set of experiments for any given scheme is to see if the performance is sensitive to variations of these external factors.

Topology

The topology of the network can be specified in the input file of the simulator. The topologies that we have chosen to examine all have high levels of redundancy. This is necessary for the success of the rerouting mechanisms, since they work by finding alternate paths from the source to the destination when the fault occurs. However, we still need to look at the effect of changing the topology on the schemes, since we would like the scheme to be general.

The first topology we picked for our initial experiments is referred to as the "mesh" topology in the rest of the thesis. It is shown in Figure 4.3. Each link in the network has a

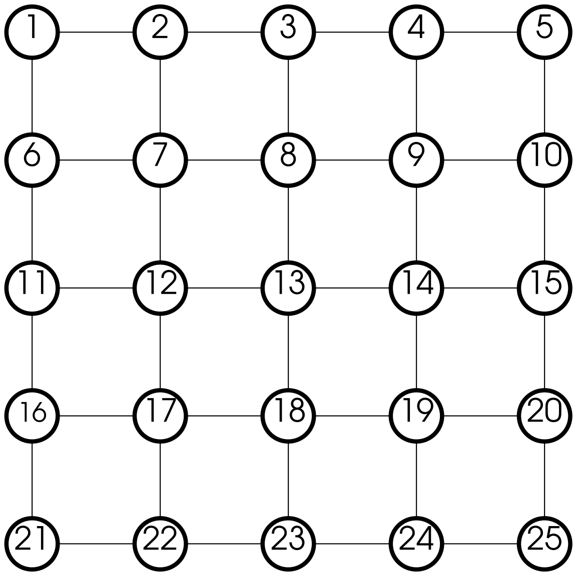

Figure 4.3: Square mesh topology

propagation latency of 10 ms, and a capacity of 10 Mbps. We chose this topology because it is simple and regular, which made it possible for us to interpret our results more easily. It also made it easier to recognize unreasonable results, facilitating the tracking down of bugs in the initial phases of our experiments. At the same time. the mesh topology has multiple disjoint paths between every pair of nodes. It is 2-edge-connected, since vertices near the corner have only two disjoint paths between them. However, vertices near the center have larger connectivity, as high as four. We considered folding the graph into a torus to make the situation symmetric, but, we felt that it would not be representative of realistic topologies. Real local and metropolitan area networks have almost planar topologies, which can be abstracted by a square mesh as we have used.

Of course, real computer networks may have links joining vertices which are not geographic neighbors, making them not perfectly planar. Thus, we chose to enhance the mesh graph with a few "trunks" to increase its connectivity and make it less regular (Figure 4.4). The trunks have a propagation latency of 25 ms and a capacity of 50 Mbps, all the other links have the same characteristics as in the mesh network. We used this as a second test topology, to check whether the change in the topology would change our

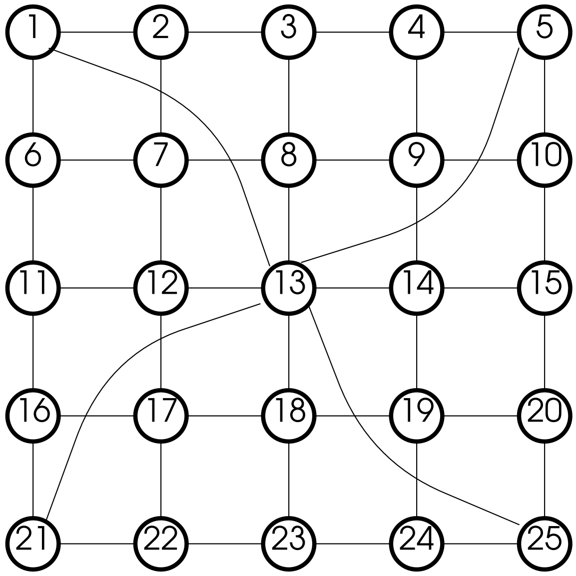

Figure 4.4: Square mesh topology with trunks

results completely. The new graph is 3-edge-connected, since the nodes in the corners have improved connectivity. We refer to this topology as the "mesh with trunks".

The two above graphs are similar, hence, in order to make sure that a completely different topology did not adversely affect our schemes, we tried one more topology based on the idea of a highly connected core, with peripheral nodes connected to the core. Many networks found in real life (such as telephone networks and airline route maps) look like this, since such topologies provide good connectivity, with short paths between any pair of nodes, at reasonable cost. Figure 4.5 shows the third network topology, "core", used in our simulations. The central core (nodes 1 to 6) is a complete clique of size six, with all links having a capacity of 10 Mbps and a latency of 10 ms. The peripheral nodes are locally gathered into two rings, and connected to three of the geographically closest central nodes to ensure redundancy. The ring is composed of links, each of 5 Mbps capacity and 10 ms latency. The links connecting the peripheral nodes to the central core are also 5 Mbps and 10 ms links. The graph is 3-node-connected, since we can find at most three node-disjoint paths between peripheral nodes in separate rings. However, we can find five *edge*-disjoint paths between any pair of nodes. Nodes within

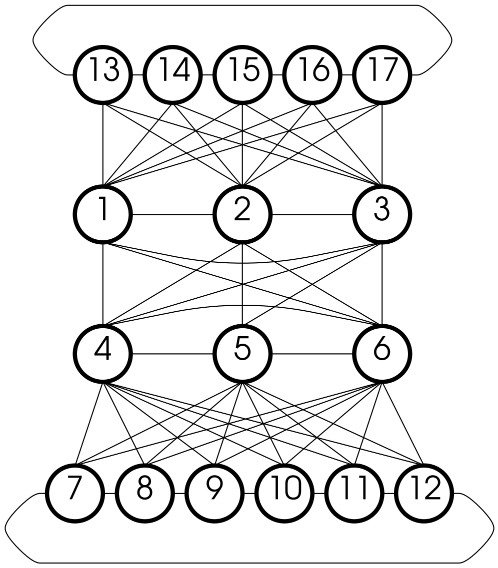

Figure 4.5: Core topology

the same ring have both a node-connectivity and a link-connectivity of five.

All our sample topologies are well connected. We must keep in mind that our schemes will not work well on poorly connected network such as trees. However, if our schemes work well on these three topologies, we can feel more secure that their success is not dependent on a particular topology, but that they will work well on any well connected network.

Traffic Mix

The nature of the applications on the network would determine the traffic parameters and the performance requirements demanded of the network. The choice of these parameters in a simulation is a difficult one, since no clear picture exists as of today of the traffic characteristics and performance requirements of applications which will run on realtime networks when they become widely available. We must make the best guesses we can about the classes of applications, their requirements, and the relative numbers of different types of applications using the network. For our experiments we chose three classes of channels: one-way JPEG-compressed medium-quality video conferencing channel: 2 Mbps (class A), CD quality audio channel: 200 Kbps (class B), and telephone quality audio channel: 64 Kbps (class C). The admission control tests used in the simulator only consider X_{min} and the delay bound d_m, so we keep $X_{ave} = X_{min}$. The value of I, therefore, is not significant. S_{max} was set to 10,000 bits for all packets, since this gave a reasonably small (1 ms) transmission time on the 10 Mbps links. X_{min} was calculated accordingly, from the rate requirements of each class.

We generated our first set of workloads from a distribution containing 30% class A channels, 30% class B channels, and the rest from class C. The source and destination of the channels were selected at random from the nodes on the periphery of the mesh. The delay requirement for the channels was generated from a uniform distribution within $(x + 50, x + 100)$ ms, where x is the one-way minimum propagation delay from the source of the channel to its destination. Thereafter, different workloads were created by using different seeds for the random number generator to get different sets of channels. The number of channels in the set was varied to get workloads with different values of the load index, as explained in the next section. The workloads generated with these distributions are referred to as the "original mix" in the rest of this chapter.

To see if the nature of the application mix had a serious impact on our simulation results, we generated another set of workloads with the following characteristics. The distribution of the channels was changed to 10% class A channels, 40% class B channels, and 50% class C channels. The distribution of source destination pairs was also changed so that channels would always go either left to right or up to down in the mesh. Finally, we changed the values of the load levels for this distribution (as explained in the next section). This distribution is referred to as the "new mix" in the results.

Load

We perform all our simulation experiments for four levels of network load, keeping all other factors constant. This allows us to see how the scheme under consideration behaves as load in the network increases. The load for the simulation is determined by the set of channels alive in the network when a fault occurs. This is determined by the set of establishment events in the event file before the fault event. We generate different loads by creating event files with different numbers of channels from the distributions defined above.

We use four load levels in each set of experiments, which we refer to as "low," "medium," "medium high," and "high". They correspond to different values of the load index defined in Section 4.2.4. The values used in one set of experiments (part of the "original mix" workload) are shown in Table 4.1. Also shown are sample utilization values from one of the workload files, to help the reader get a better feel for the meaning of the load index in terms of a more familiar metric of load. The average utilization is the total reserved rate expressed as a percentage of link capacity averaged over all links in the network after the establishment of all the channels in the workload file, while the maximum is the utilization on the most heavily loaded link. We must, however, emphasize that the load index captures more information than is captured by utilization, since a set of channels with identical rate parameters but different delay requirements would have the same utilization values but different load indices.

level	load index	average utilization	max utilization
low	270	15.8 %	63.3 %
medium	616	32.0 %	84.7 %
medium-high	777	41.2 %	87.3 %
high	858	44.5 %	86.7 %

Table 4.1: Load index levels

The different network topologies have different capacities; thus, different values of the load index were used in experiments involving different topologies. In addition, for the "new mix" distribution we also increased the load values to see if that would change the behavior of our schemes. Thus, while each set of experiments has four load levels, the values may differ and will be printed on the load axis of each graph.

Each set of experiments was conducted with twelve workload files, consisting of three workload files at each of the four load levels. The three workload files at a given level were generated with the number of channels picked to give a load value close to the desired load index when all the channels were set up. The results from each set of three were averaged to increase our confidence in the results. Each of these experiments takes an hour or more to run, depending on the class of workstation the simulator is running on. Moreover, each experiment had to be repeated across a large number of external factors and internal factors (described below). Thus, conducting a significantly larger number of experiments at each load level, while desirable, was not feasible.

We are using the results to compare the performances of schemes. The numerical values obtained as results are specific to the network topology and load mix. In addition, the numerical values depend very much on the specific set of channels to be rerouted, so that even at the same load level they vary widely. However, some clear trends are observable about the *relative* performance of the schemes. Thus, even though the numerical values do not converge, some schemes perform consistently better than others. This is the kind of result that we shall present in the simulation section. Since we did not perform each experiment a large number of times, statistical tests of confidence could not be conducted.

Failure model

The nature of the failure being simulated has an influence on the ability of the schemes to recover from the failure. We place certain restrictions on the failures being simulated. We only deal with link failures.[2] We also deal only with *fail-stop* failures, in which the failed component does not continue to transmit bad messages into the rest of the network. *Byzantine* failures of nodes are especially hard to deal with, since it may not even be possible to detect and isolate the failure. In the simulations performed, when a

[2] Node failures may be dealt within this model by treating them as simultaneous multiple failures of all links connecting to the failed node. However, for simplicity we restricted the scope of the simulations performed to link failures.

link fails, it simply stops transmitting data.

Finally, multiple failures are harder to deal with than single failures, because more of the network load is affected, and less of the network is available to service the recovery. Multiple faults also aggravate the problem of collisions, discussed in Section 3.5.4. In addition, the physical distance between the two failures, and thus, between the two sets of channels being rerouted, would affect the timing behavior of the route update process.

Some links of wide-area computer networks may be leased lines, which are themselves routed on top of the cross-connect layer of a telecommunication network. Thus, it is possible for a single physical failure to be manifested as the multiple failures of two or more links of the computer network, unless care is taken to place logically different links (at the higher layers) in physically separate links.

We simulate single faults and double faults in our experiments. However, since a single fault is more likely, we consider recovery performance for the single fault cases more important than that in the double fault cases.

4.2.6.2. Internal factors

As described in the previous chapter, the internal factors of the recovery scheme are: locus of reroute, timing, retries, and state prediction. The simulator is programmed to accept parameters from an input file, which allows the user to select combinations of values for the internal factors of the scheme. We will experiment with the effect of the internal factors on the performance, to identify how much of an effect each factor has on it, to select the correct approach for the important factors, and to identify interactions between factors. The objective of these experiments is to come up with a scheme that provides good performance on all the metrics of comparison and is tolerant to variations in the external factors.

4.3. Results

In this section, we discuss the most interesting results of the simulation experiments performed using the framework described above. We first present the results concerning the effects of the internal factors, choosing as levels of the external factors the "mesh" topology, the "original mix", and single failures. Results are shown for all four load levels. The next four subsections show the effects of the four internal factors (timing, routing constraints, retries, and state prediction).

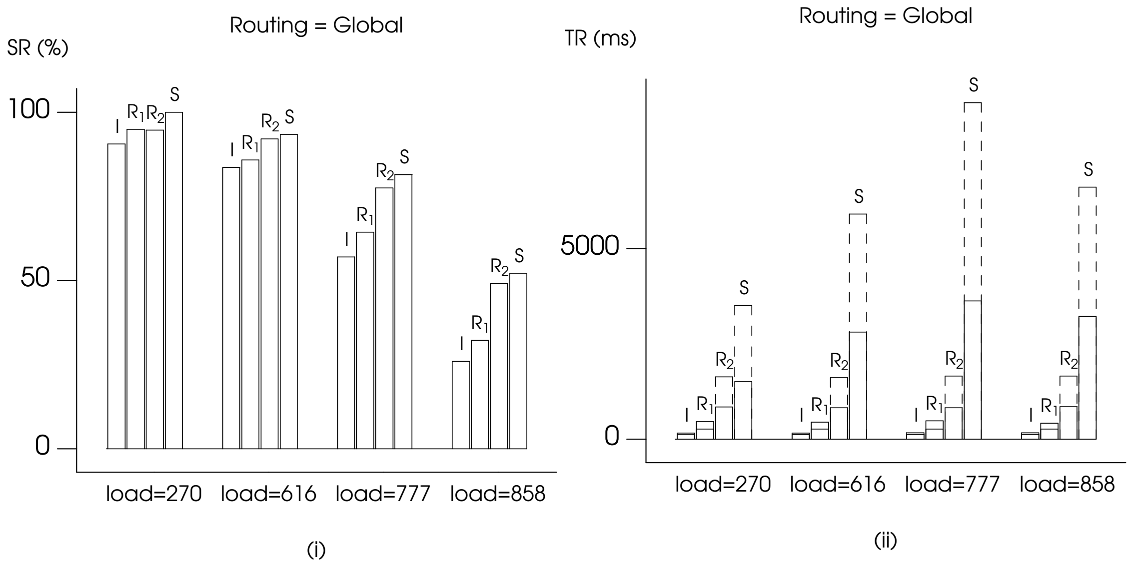

Figure 4.6: Effect of timing on (i) reroute success (ii) time to reroute

4.3.1. Effect of timing

Figure 4.6 shows the effect of varying the timing, with the other internal factors fixed (Global locus of reroute, no retries, and Local prediction). The timing is varied across Immediate (I), Random over 300 ms (R_1), Random over 1500 ms (R_2), and Sequential (S). Graph (i) shows the reroute success in terms of the Success Ratio (SR) defined in Equation 4.6. Graph (ii) shows the time to reroute (TR). The lower solid box shows the average time to reroute, averaged over the set of successfully rerouted channels, measured from the time of the fault to the time of successful re-establishment. The upper dashed box shows the time to reroute for the last channel to be successfully rerouted.

We observe that, while Sequential successfully reroutes far more realtime load than Immediate (roughly twice as much under high network loads), the time to reroute for Sequential rules it out as a practical scheme. However, as we observed before, it is a useful control scheme to compare the reroute success of other more practical schemes. Ideally, we would like a scheme with the success of Sequential, but the timing characteristics of Immediate. Randomization achieves this goal to a limited extent, since we reroute some fraction of the difference between Immediate and Sequential at a more reasonable time to reroute. However, there is scope for improvement, both in terms of the success ratio and the time to reroute.

The average time to reroute a realtime channel using Immediate timing is about 130 ms. The average path length (from the distribution of source-destination pairs that we used for this experiment) is five. Since each link has a propagation latency of 10 ms, the expected round-trip time would be close to 100 ms. The remaining time can be attributed to the time for the fault message to reach the source node from the location of the fault. The maximum time to reroute for the Random timing case is close to $n + 150$ ms, where n is the randomization interval, which is as expected. The timing characteristics of Immediate and Random are independent of the load, since they are determined by the time taken to perform a round trip, and the value of the randomization interval. However, the time to reroute for Sequential changes with the load, since it depends on the number of channels being rerouted. It increases initially, as more channels are being affected by the fault as the load increases. However, at high load it drops off, since less channels can be successfully rerouted, and the maximum and average are calculated over the set of successfully rerouted channels.

We only show the results for Random over 300 ms and Random over 1500 ms. Other values of the randomization interval follow the same trends.

--

4.3.2. Effect of locus of reroute

Figure 4.7 (i) shows the effect of the locus of reroute on the reroute success. The timing is fixed to Random over 300 ms. The results for other timing values follow a similar trend. We see that Local reroutes a far smaller set of channels than either Global or Hybrid, especially at higher loads. This can be explained as due to the limited choice in routes, since Global rerouting looks for a new route from the source to the destination, while Local rerouting only tries to replace the section of the route corresponding to the failed link. Thus, the set of possibilities considered by Global is much larger than that considered by Local. This reduces the chances of Local

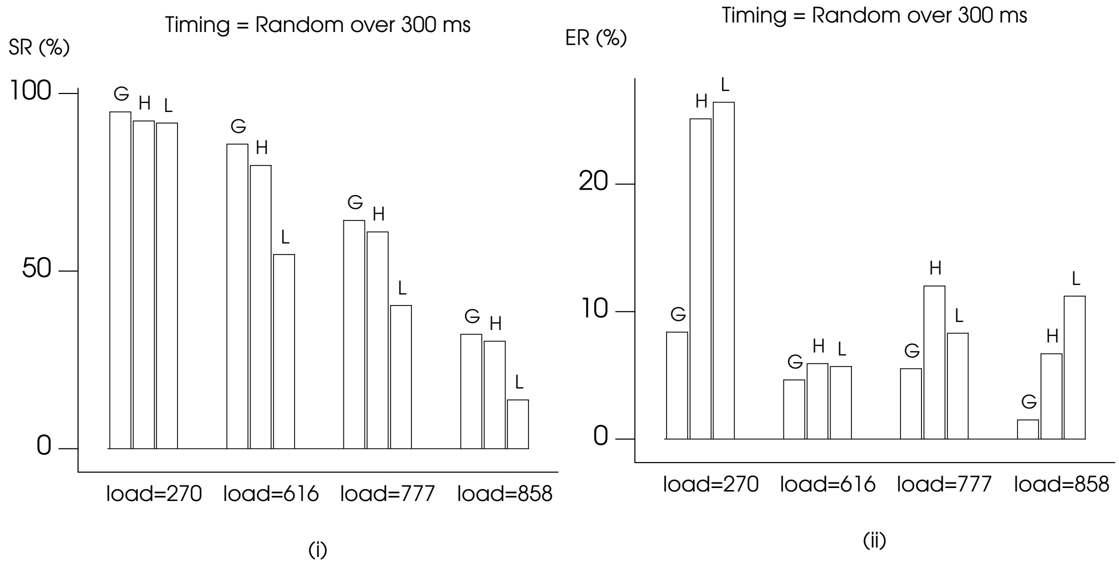

Figure 4.7: Effect of locus of reroute on (i) reroute success (ii) excess resources used

finding a route in heavily loaded networks, as well as makes it likely that any route found by Local will be longer than one found by Global. The performances of Global and Hybrid are comparable, since, during Hybrid rerouting, any time the Local route cannot be found, a Global route is sought. On the basis of these results, we have a strong reason to reject Local rerouting as the correct approach to locus of reroute, but have no clear choice between Global and Hybrid.

The effect of the locus of reroute on the efficiency of resource usage (ER) is shown in graph (ii). Global rerouting is more efficient than the other routing schemes, across all load values. This result holds across the other timing schemes too, and is mainly because Global tries to find the shortest existing path from the source to the destination. This goal also minimizes resource usage. Thus, from the point of view of resource usage, the performance of Global rerouting is best.

Changing the locus of reroute does not change the time to reroute; hence, we do not show a graph for it. The time to reroute for Local and Hybrid are very similar to that seen in Figure 4.6 (ii), for all the approaches to the timing policy. For example, for Immediate timing, the average time to reroute is close to 130 ms, regardless of the locus of reroute approach chosen. This is because the time to reroute is dominated by the round-trip time for establishment, and the random waiting time in the case of the Random approaches. Since the round-trip time is only marginally affected by the locus of reroute, and the waiting time not at all, the overall effect of the locus of reroute on the timing is negligible.

4.3.3. Effect of retry policy

The effect of the retry policy on the performance metrics is shown in Figure 4.8. Graphs (i) and (ii) show the success ratios for Immediate retries and Delayed retries. We fix the locus of reroute to Global and look at the performance across Immediate, Random over 1500 ms, and Sequential. At each load level we show six bars. The dotted bars depict the performance for the scheme with retries, the plain bars show the performance without retries. The performance for Immediate retries, with up to four retries in the event of failure to reroute, is shown in graph (i). As we can see, the success ratio improves when we add retries, but the improvement is smaller than the improvement achievable through randomization. Increasing the number of immediate retries does not appreciably improve the success ratio.

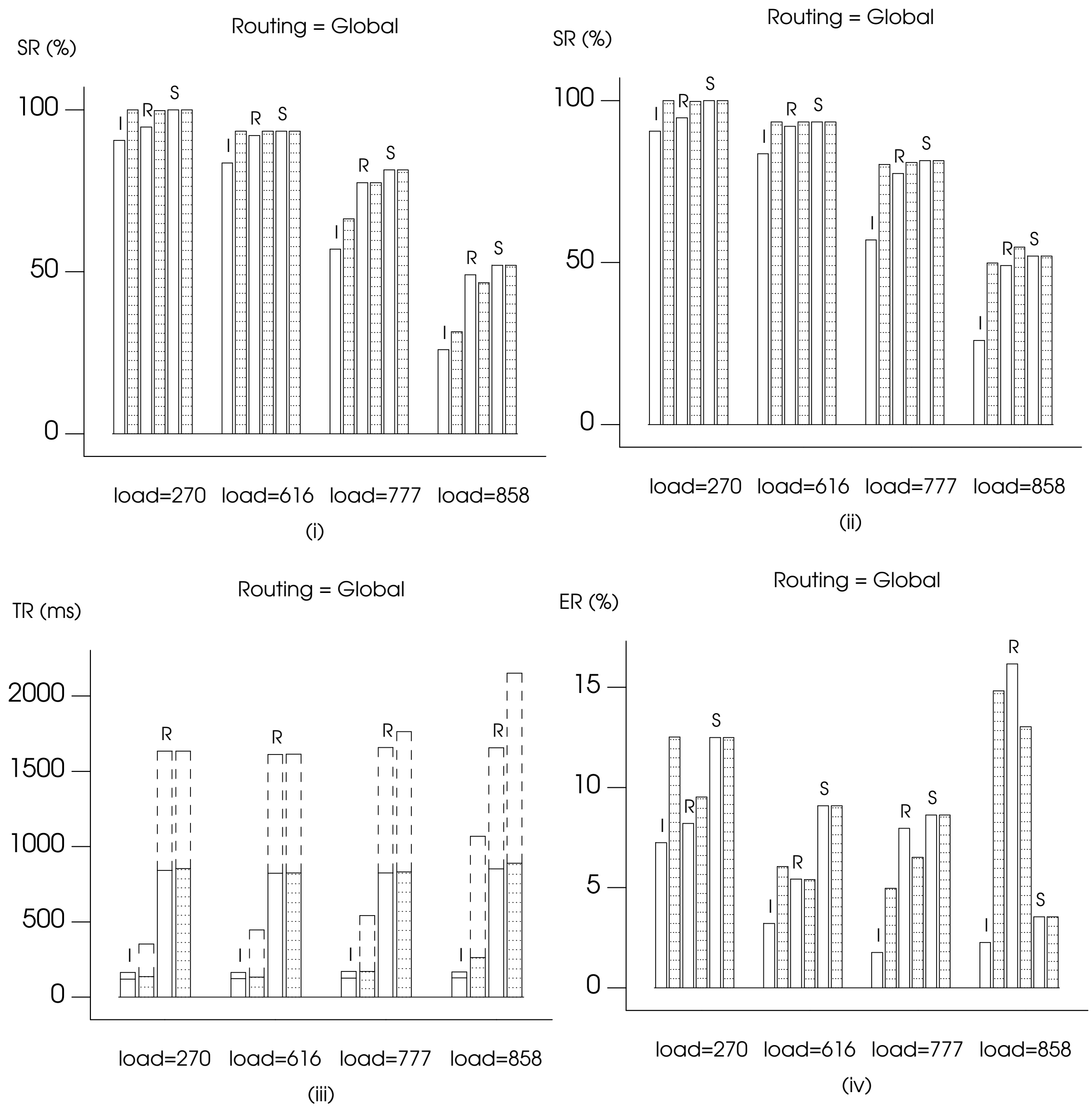

Figure 4.8: Effect of retry policy. (i) Immediate retries (ii)- (iv) Delayed retries

The success ratio for Delayed retries is shown in graph (ii), where we can see that this approach is much more successful. In fact, Delayed retries can reroute as much realtime load as Sequential timing, which is about the best we can do assuming a greedy strategy, with the same routing algorithm and locus of reroute constraints.

The performance of Delayed retries, as regards the time to reroute metric, is shown in graph (iii). We only show the time to reroute for Immediate and Random, with and without Delayed retries, since the time to reroute of Sequential is known to be very poor (Fig. 4.6, (ii)), and plotting it would make the graph harder to read, because the scale for Sequential is much larger. We note that the Immediate timing scheme, when we add retries, suffers from an increase in the average time to reroute, and a much larger increase in the maximum time to reroute. The retry policy used to generate the graphs shown is based on an exponentially increasing period[3] over which the random delay is chosen. Thus, the later retries are likely to occur after longer waiting periods.

From graph (iv) we can see that adding retries causes the resource usage to go up, but this is reasonable, since the amount of realtime load rerouted is also increasing. The resource usage for Global does not exceed 15% over the minimum required. The performance of Global in this regard continues to be better than that of the other schemes.

The time to reroute can be improved significantly without loss of reroute success or of efficiency of resource usage, if we fix the retry interval to the *correct* value, instead of starting from a small interval and exponentially increasing it. Picking the right value for the initial retry interval can make sure that most of the reroutes never go into a third retry. For the "mesh" network this value was experimentally found to be around 100 ms. This is also the expected round-trip time for the traffic mix chosen. This can be explained as follows: if the initial attempt failed due to a collision with another establishment in the network, the first channel should wait for the establishment to finish and for the route updates generated by the establishment to reach the source node (of the first channel). The channel establishment will take on average half a round trip time to complete, since on average it will be half completed when the collision happens. The information from the farthest node will take on average half a round-trip time more to get to the source node. Thus, we should generate our retry at this time.

Figure 4.9 shows the change in the histogram of successfully rerouted channels

[3] The range from which the random delay is chosen starts from a small value, and is doubled for each additional retry.

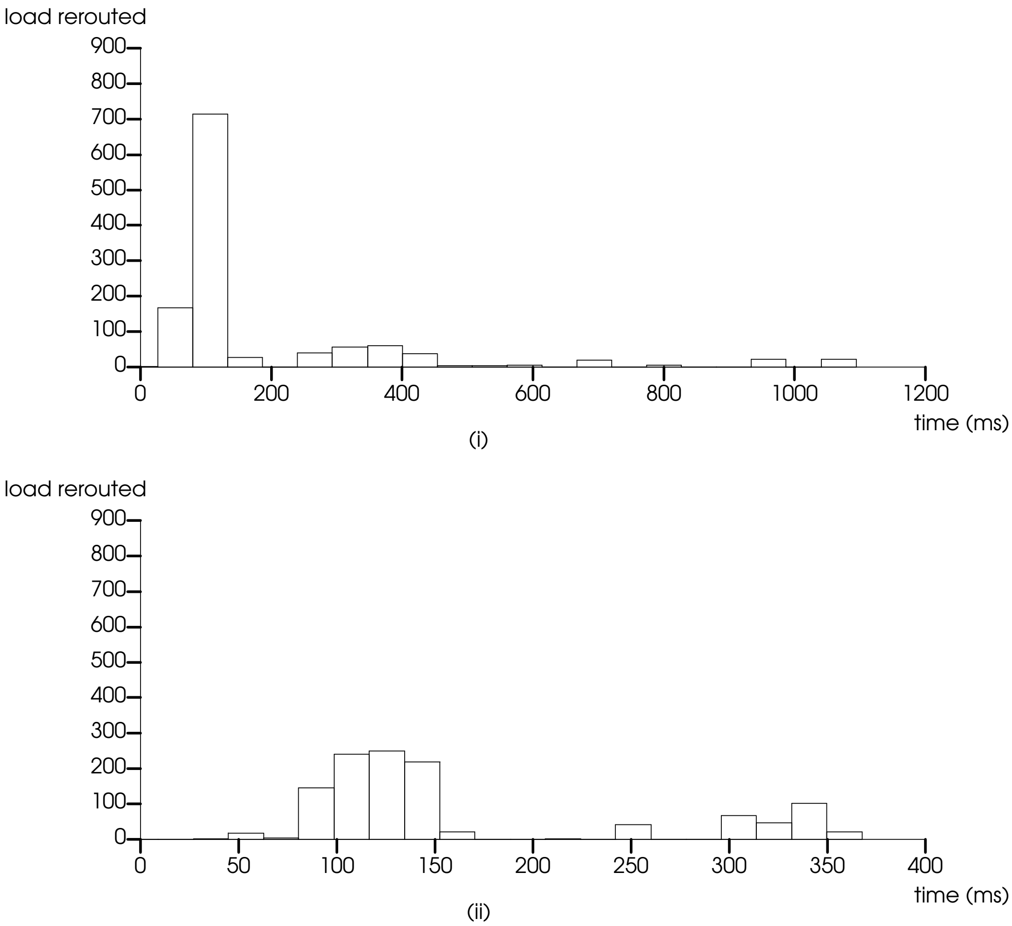

Figure 4.9: Effect of retry interval on the histogram of time to reroute (i) Exponential (ii) Fixed (100 ms)

(amount of realtime load vs. time to reroute), when we fix the retry interval to 100 ms. This was the best performance obtained with regard to the time to reroute metric out of a large number of experiments with different methods of selecting the delay for each retry, such as different fixed intervals, randomly chosen intervals, and increasing intervals. Picking the right retry interval can pull in the tail of the histogram dramatically. In general, the correct value of the retry interval will depend on the network topology, and the distribution of path lengths. Our hypothesis is that the average round-trip time is a good estimate of the value the retry interval should be set to. This hypothesis will be tested on a different topology in Section 4.3.7.

The performance is sensitive to the selection of the correct interval. Thus, this should be a tunable parameter of any implementation, to allow the value to be correctly selected for a particular network topology and channel path-length distribution. Also, evaluation of the variation of performance with respect to this parameter, in the context of a real implementation, would be valuable.

We also experimented with different numbers of retries, as well as using a threshold of time to halt the retry process. These allow the maximum time to reroute to be selected to adjust for different application requirements; any channel remaining after the retry process is halted is torn down and its user informed. While the number of retries and the threshold method have very similar performances, the latter appears to be a more natural way to specify the maximum recovery time needed from the network.

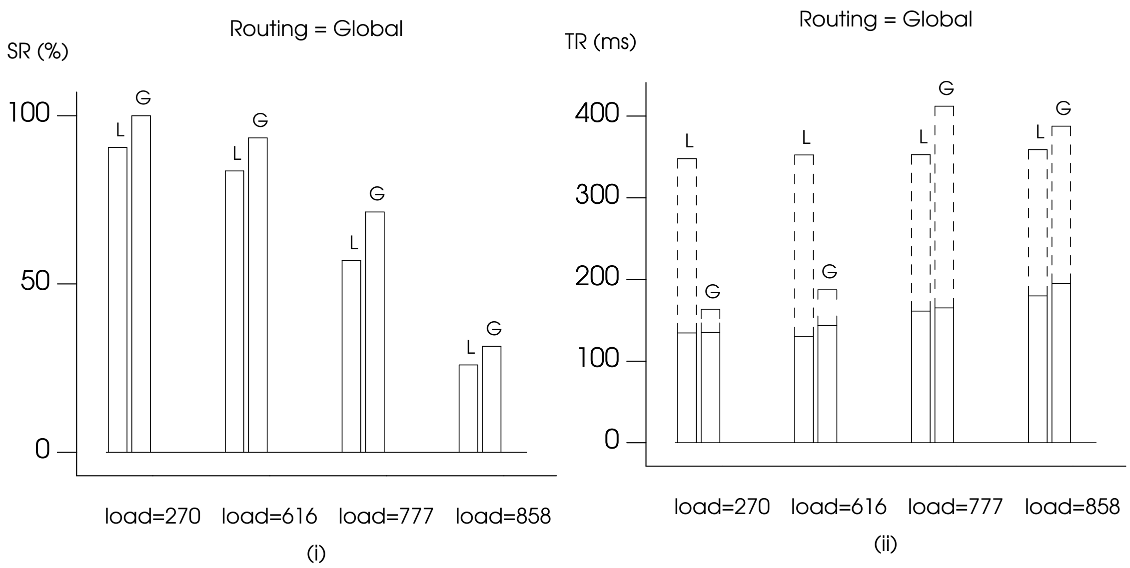

Figure 4.10: Effect of state prediction on
(i) time to reroute (ii) realtime load rerouted on first attempt

4.3.4. Effect of state prediction

Global prediction attempts to use knowledge of the behavior of other nodes in the network to place reroute attempts on paths where collisions will not occur. This improves the rerouting success on the first attempt, making the retry policy less important. This can be seen in Figure 4.10, graph (i), where the amount of realtime load successfully rerouted in the first attempt (without retries) is plotted. The two bars shown for each workload are "Local prediction" (L) and "Global prediction" (G). With Global prediction, the first attempt always reroutes more realtime load, leading to quicker recovery times for more clients.

However, this amount of realtime load is less than that recovered by the Immediate timing with Delayed retries scheme (the first dotted bar in Figure 4.8 (ii)). If we use Global prediction in conjunction with Delayed retries we can recover exactly the same amount of realtime load. However, the timing of the aggregate scheme is determined by the Delayed retry process, as shown in Figure 4.10, graph (ii). Both schemes use retries set to an interval of 100 ms, which we saw performed well. We see that Global prediction performs slightly better at low loads and comparably, or slightly worse, at high loads. The maximum time to reroute is affected more than the average. It seems that, as long as the load is low enough to allow the prediction algorithm to reroute all the realtime load in the first synchronized rerouting attempt, we get better timing with the Global prediction scheme. However, as soon as the load increases to the point where the scheme is forced to use retries, the timing is slightly worse than with the Local prediction scheme.

We can also see this in the histogram of the time to reroute (Figure 4.11). The histogram for the scheme with Global prediction, shown in graph (ii), has a larger area under 200 ms. However, because of the need to synchronize the reroute time across all the nodes, the scheme with Global prediction adds an extra 50 ms or so to the start of the recovery process. This shows up in the histograms as well as in the maximum time to reroute.

One element that is ignored in our simulations is the computation time to run the algorithms. We justified this in Section 4.2.1 by looking at the measured values of the computation time. However, that argument does not hold for Global prediction, since the

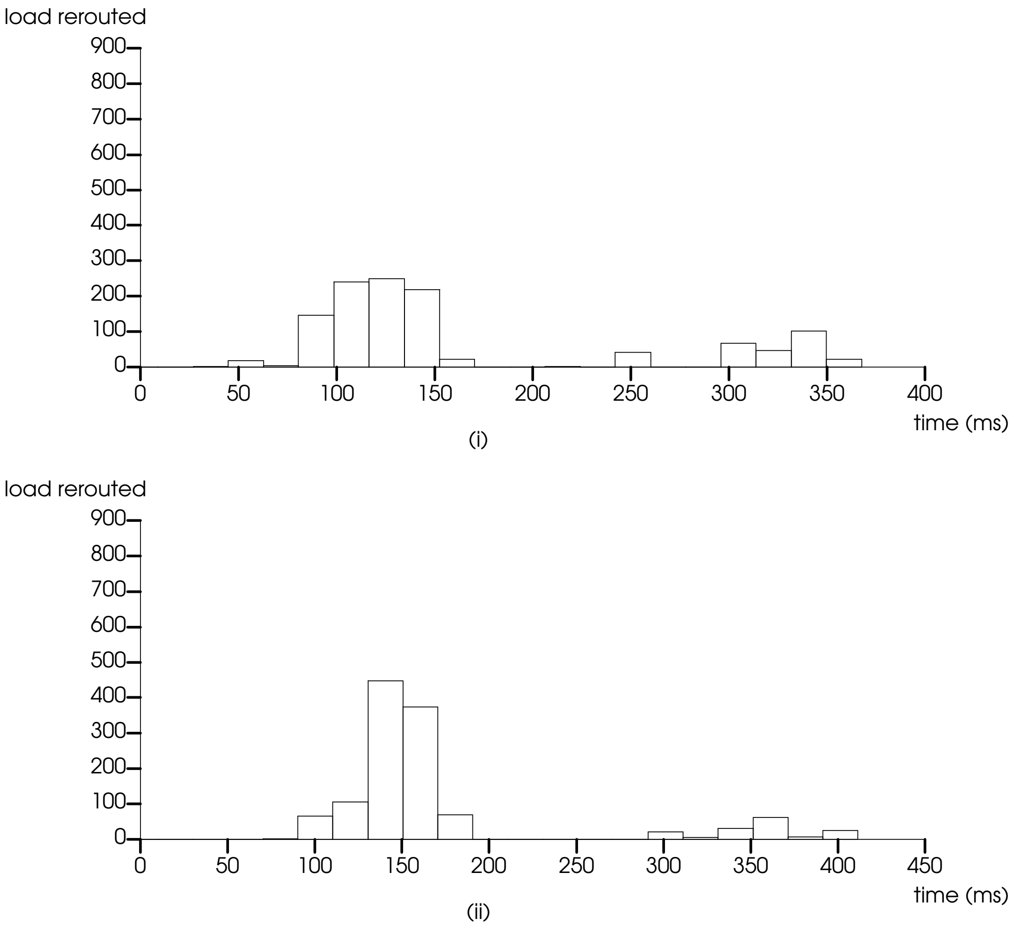

Figure 4.11: Effect of state prediction on histogram of time to reroute (i) Local prediction (ii) Global prediction

algorithm involves running the routing algorithm for all the affected channels at each node. The workloads being simulated have at most twenty channels affected by a single fault. Since the synchronized algorithm is run only once at the start of the recovery (if the retries are not using Global prediction, as was the case in the experiment that generated graph (ii)), we can approximate the effect of including this computation time by sliding the histogram to the right (increasing the recovery times) by 20 ms. This does not have a big effect on our results; however, it does make the Global prediction scheme, which already has a slightly higher maximum time to reroute than the Local prediction scheme, even less attractive.

Overall, it appears that, though Global prediction does improve the amount of real-time load rerouted in the first attempt, even more can be rerouted by using retries, and the timing of the Global prediction scheme with retries is dominated by the timing of the retry process. In addition, the extra overhead incurred by the Global prediction scheme at the start shows up in the form of a slightly higher maximum time to reroute.

Moreover, we note that the synchronization overhead to get all the nodes to start the establishments at the same time, as well as the extra computation time to run the Global prediction algorithm, would scale with the number of channels affected by the fault, and with the size of the network. For small networks, the Local prediction scheme seems to work already slightly better than the Global prediction scheme, and this difference should tend to increase with the size of the network. Thus, overall, the gains offered by Global prediction do not seem to justify the overheads and the increased complexity of the scheme.

4.3.5. Effect of number of faults

The (Global routing, Immediate timing, Delayed retry, Local prediction) scheme with a well-chosen retry interval works well in the case of a single fault. To see whether it continues to work for multiple failures, we simulated two faults in the network. Figure 4.12 (i) shows the success ratio for the Immediate (I), Immediate with Delayed retries(ID), and Sequential (S) schemes. The Delayed retry scheme works as well in the double failure case, leading to success ratios as high as in the case of Sequential timing. Graph (ii) compares the performance of the Global prediction scheme with that of the Local prediction scheme, by showing the amount of realtime load rerouted in the first attempt (i.e., with no retries). As before, the Global prediction scheme performs slightly better in the first attempt by rerouting slightly more realtime load. Graph (iii) compares the effect of two failures on average and maximum time to reroute. The two bars shown for each load are the Immediate without retries, and Immediate with Delayed retries set at 100 ms intervals, both with Local prediction. The timing of the Global prediction scheme with

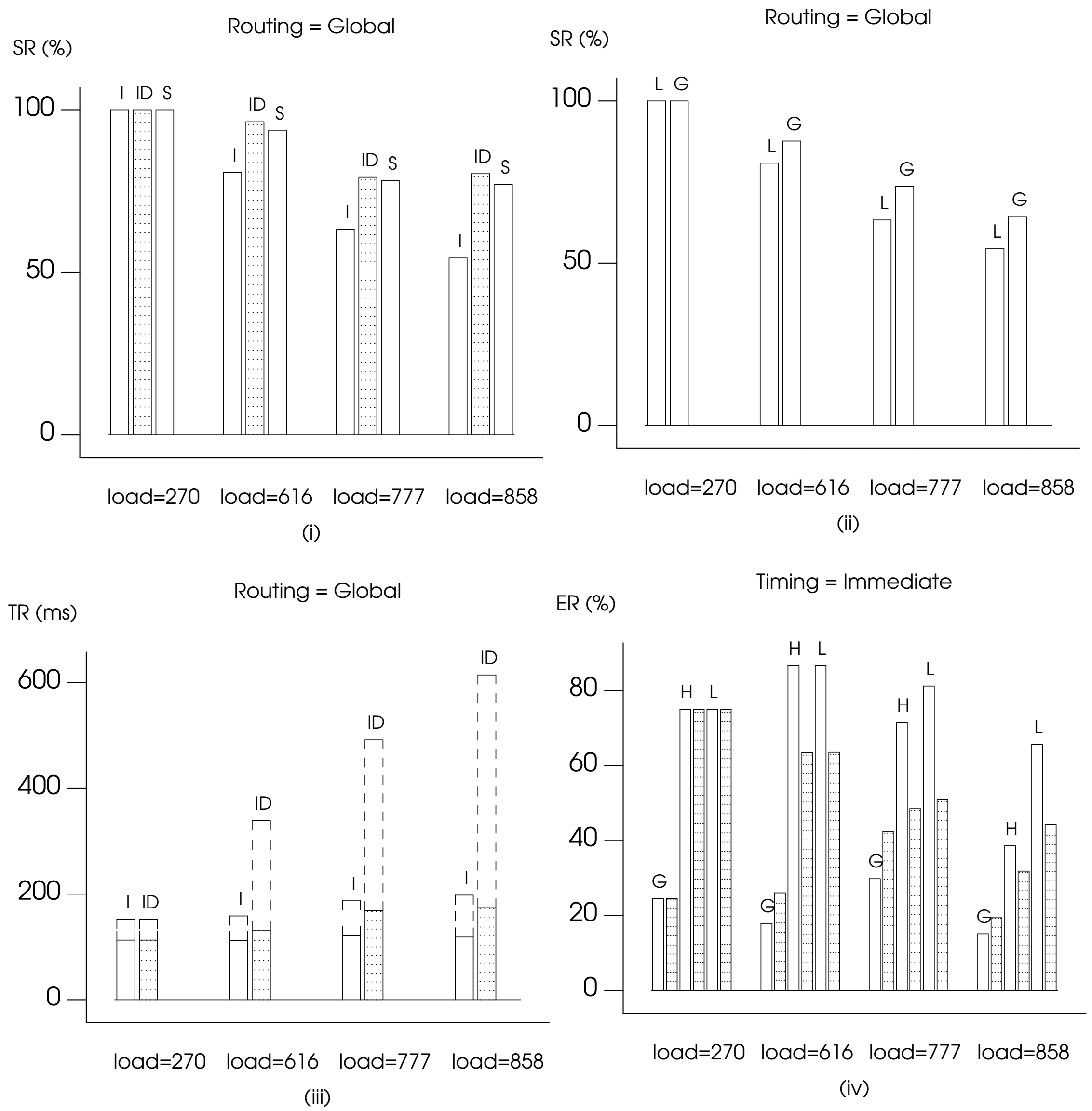

Figure 4.12: Effect of number of faults on
(i) reroute success (ii) success of prediction (iii) time to reroute (iv) excess resources used

retries is very similar that of the scheme with Local prediction (graph (iii)), and hence is not shown. The maximum time to reroute is longer for the two-failure case, since in Figure 4.9 we saw that the single failure maximum recovery time was less than 400 ms. However, the two failure experiment affected a larger number of channels; hence, a longer time to recover is acceptable. We experimented with different values of the retry interval, but the best performance, shown here, is still for retries at 100 ms.

Graph (iv) looks at the excess resource (ER) used. We can see that Global rerouting continues to perform well in this respect, but Hybrid and Local perform worse. The excess resource used decreases when adding retries for Hybrid and Local locus of rerouting. This is an artifact of the way in which retries are implemented; all retries are performed using Global locus of reroutes. Thus, in the presence of retries, the scheme tends to behave more like Global, and the efficiency of resource usage improves. This supports our earlier observation in favor of Global locus of reroute.

In general, the following conclusions are valid for both the single and double failure experiments:

- Immediate rerouting has the best time to recover.
- With Delayed retries the success ratio of Immediate retry can be brought up to match that of Sequential rerouting.
- By appropriately choosing the retry interval we can keep the time to reroute down to practical levels (average 200 ms, maximum 600 ms for two failures at high network load).
- Global locus of reroute is the most efficient in terms of resource utilization.

4.3.6. Sensitivity to load mix

Figure 4.13 shows the effect of a different load mix on the performance of some of the approaches to failure recovery. The load mix used to generate this graph is the "new mix" described in Section 4.2.6.1. The load levels used in this mix are shown on the x-axis of the graphs. They span a wider range of values than the ones used in the previous mix. The mix also uses different ratios for the different classes of traffic, and a different distribution of (source, destination) pairs.

The three bars shown at each load level in graph (i) are the success ratios for Immediate timing without retries (I), with Delayed retries(ID), and for Sequential timing(S). Adding Delayed retries to Immediate timing allows it to recover as much realtime load as

--

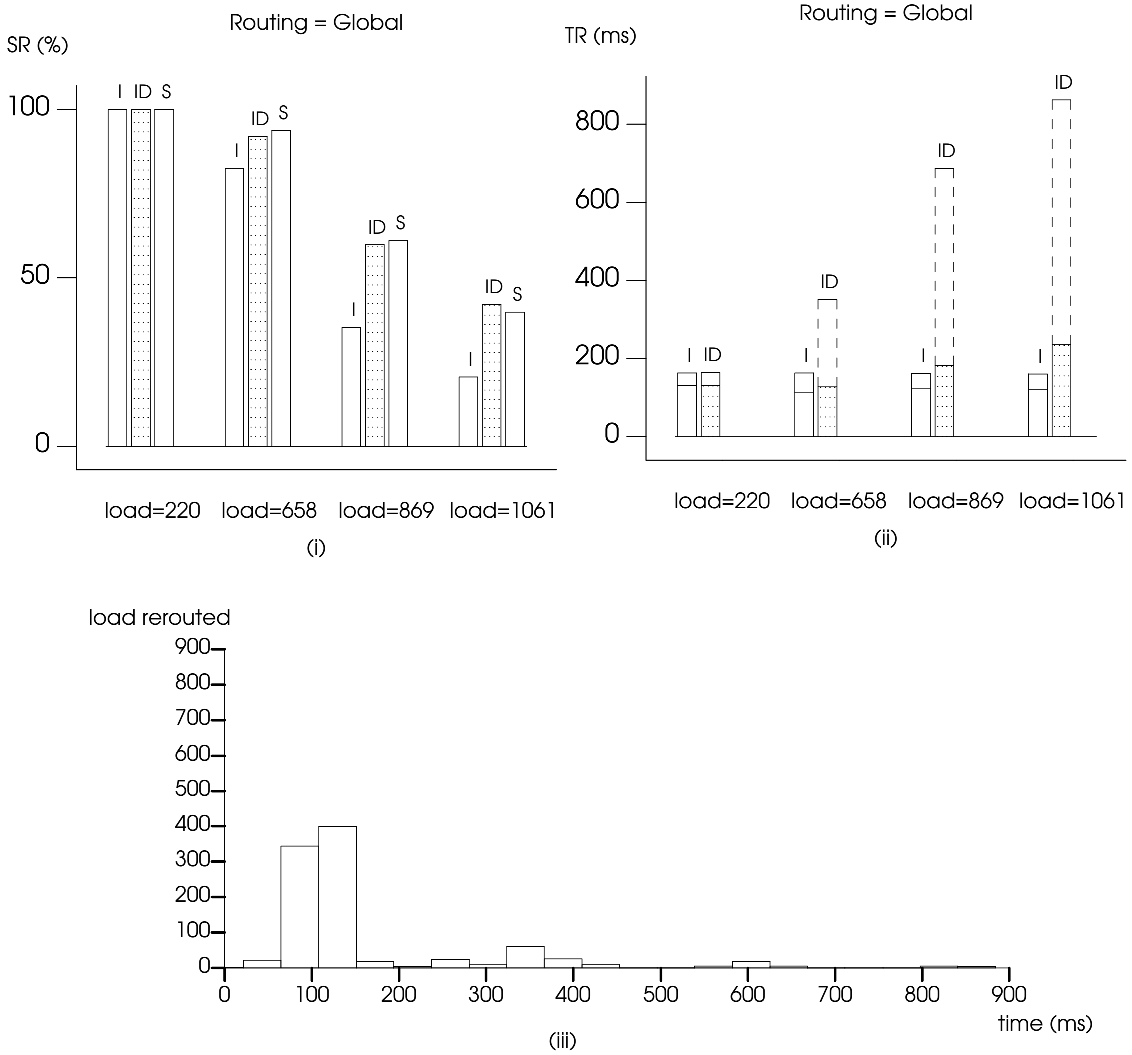

Figure 4.13: Effect of new load mix on
(i) Success ratio (ii) Time to reroute (iii) Histogram of time to reroute

the Sequential approach. At the same time, graph (ii) shows that the time to recover is not too long. The plain bar shows the time to recover for the Immediate timing scheme, while the dotted bar shows the effect of adding retries. The load at the highest level in this mix is more than in the last mix, so the recovery schemes are stressed more. This increases the maximum time to recover as compared to that shown in Figure 4.9 or Figure 4.12. The average is less affected. The histogram of time to recover for the Delayed retry scheme is shown in graph (iii). We can see that very little realtime load lies in the tail of the histogram; hence, most clients would see delays less than 400 ms. This is a desirable characteristic of the Delayed retry scheme. We can also easily trade the amount of real-time load rerouted for the maximum time to reroute, to accommodate tighter response requirements, by not generating retries beyond a certain time after the fault.

To summarize, the scheme with Delayed retries adapted well to the change in load mix. It continued to perform as well as the Sequential scheme in terms of the reroute success. The maximum time to reroute increased; this can be attributed to the increased load levels in the network. The change was not catastrophic, but graceful, since the average time to reroute did not shift much, and few channels were affected by the increased time to reroute. The efficiency of resource usage (not shown here) was also not changed significantly.

4.3.7. Sensitivity to topology

Figure 4.14 shows some of the results of the experiments we performed to verify the effect of changing the topology on the recovery schemes. In graph (i) and (ii) we look at the effect of adding some trunks to the mesh topology to yield the "mesh with trunks" topology shown in Figure 4.4, using the "original" load mix. Graph (i) shows the effect on the reroute success. Adding the trunks increases the capacity of the network, so that, with the same sets of channels, the network is less heavily loaded. This is seen in the lower values of the load index, shown on the x-axis of the graph. Thus, the Sequential timing scheme and the Immediate timing with Delayed retries can recover all of the affected realtime load in this case. Graph (ii) shows the time to recover for the Immediate scheme without retries (plain bars) and with Delayed retries (dotted bars).

We get similar results with the "core" topology (graphs (iii) and (iv)). With the given load mix and load levels, all of the affected realtime load is successfully rerouted by the Immediate timing scheme with retries within a reasonable time. However, the time to reroute can be further improved by changing the timing of the Delayed retry. The best timing comes for intervals close to 60 ms. We can reroute the same amount of realtime load in a shorter time, as shown in the histograms in Figure 4.15. Again, this is roughly the

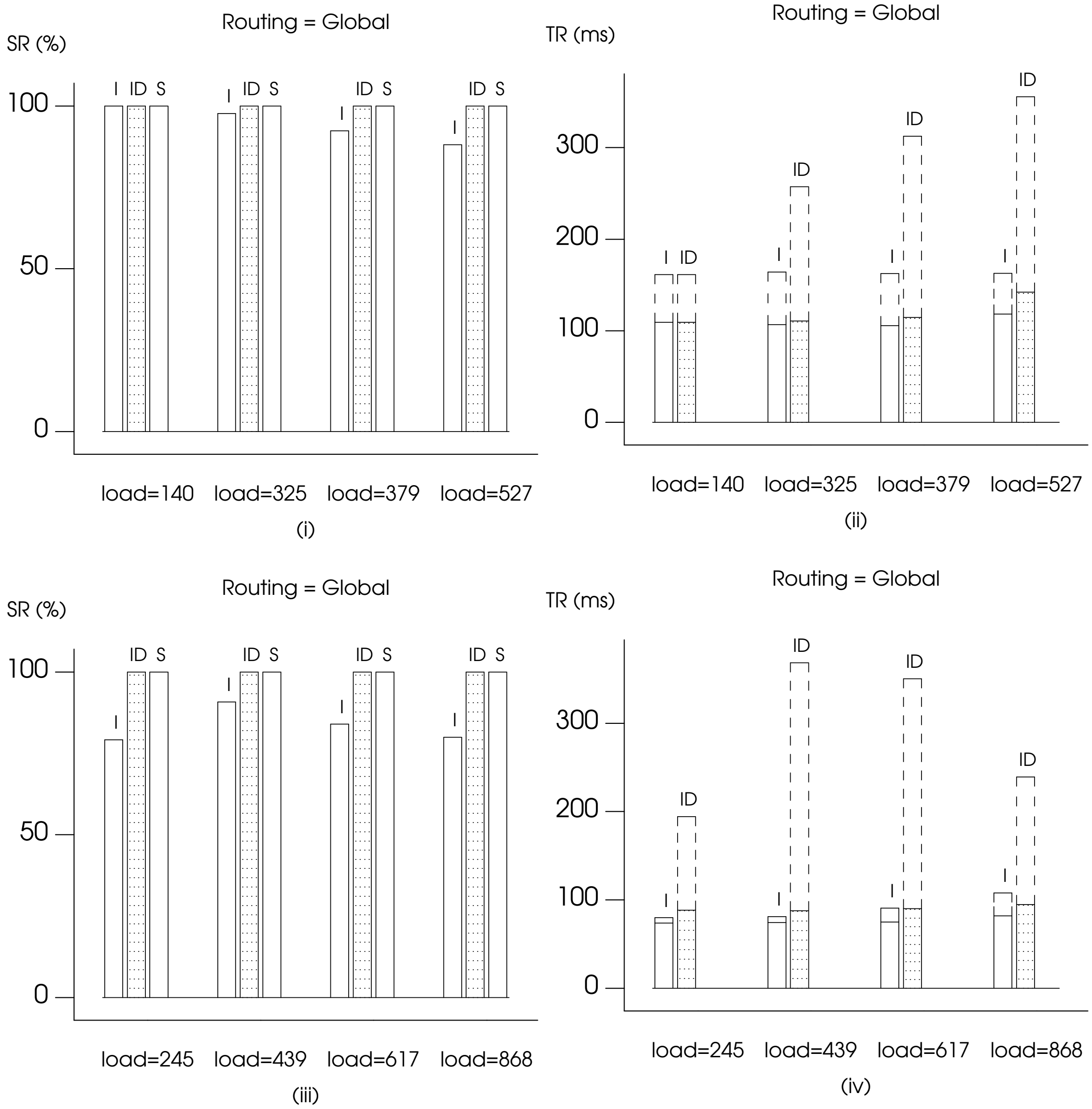

Figure 4.14: Effect of topology. (i)-(ii) Mesh with added trunks (iii)-(iv) Core topology

average round-trip time for this network, supporting our earlier hypothesis.

Thus, all the general observations made with the mesh topology continue to hold for the other topologies. The (Global routing, Immediate timing, Delayed retries, Local prediction) scheme continues to give good performance on all metrics of performance. It is quite insensitive to changes in topology, as long as the topology is well connected.

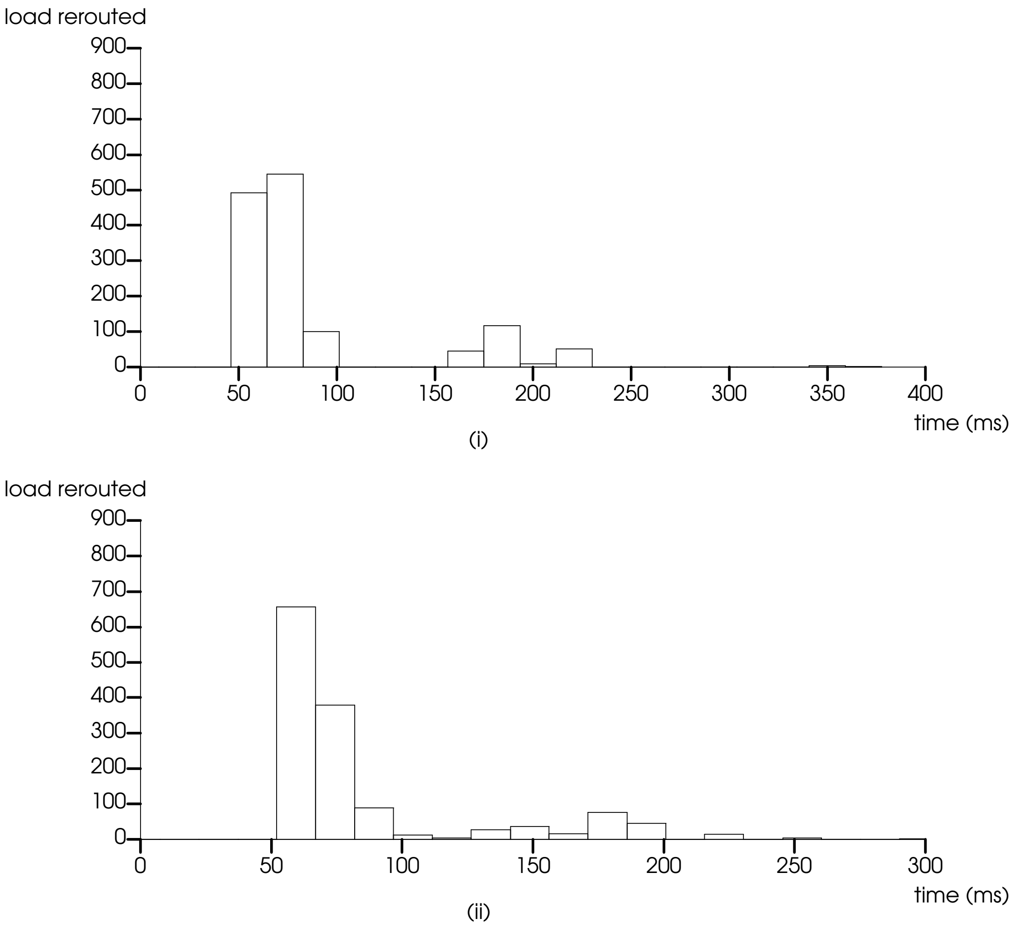

Figure 4.15: Effect of retry interval on core topology (i) 100 ms interval (ii) 60 ms interval

4.4. Conclusion

The main conclusion that we can draw from the results presented above is that schemes within the distributed, dynamic framework selected in the previous chapter can reroute a significant portion of the affected realtime load in the event of a link fault, within reasonable times and without wasting network resources excessively. The schemes are also reasonably tolerant of changes in load, traffic mix, number of faults, and network topology.

The experiments described in this chapter explored realtime channel recovery along the dimensions of reroute timing, locus of reroute constraints, retries, and state prediction. The effects of changes in these internal factors were evaluated on the three chosen metrics of recovery performance: amount of realtime load rerouted, time taken to reroute, and efficiency of resource usage.

Of the internal factors considered, the retry policy has the most profound effect on the first two metrics: fraction of realtime load rerouted and time to reroute. While Immediate retries did not provide a significant improvement in amount of realtime load rerouted, by adding a delay before initiating the retry and computing the route based on route update information that arrived in the interval, we can improve the success of the scheme to the point where it rivals the amount of realtime load rerouted by the Sequential timing scheme. As pointed out in Section 3.7.2, the Sequential timing scheme eliminates all collisions at the expense of longer times to reroute the channels. Thus, we cannot expect to reroute a significantly higher amount of realtime load using a greedy strategy, where each channel is rerouted independently of the others.

By experimenting with the interval between retries, we found an empirical value of 100 ms, at which the scheme performed best for the "mesh" topology. This is also roughly the average round-trip time given the topology and the distribution of (source, destination) pairs. This fits the theory that a significant number of unsuccessful establishment attempts occur due to collisions with other attempts during the forward pass. Thus, if the retry is timed to occur shortly after the other establishment completes *and* the information reaches back to the node where the route computation is being performed, it has the best chance of success and the shortest waiting time. However, the value of this interval will, in general, change with the network topology. A rule of thumb to help select this interval for an implementation is to set it to the average round-trip time. Since the performance is quite sensitive to this parameter, this should, perhaps, be evaluated more accurately in the context of a real implementation. In any case, the retry interval should be a tunable parameter of an implementation, to make it easy to make it adapt it to different topologies.

The locus of reroute policy has a significant impact on the reroute success and efficiency of resource usage. Local rerouting, which is similar to *link rerouting* used in telecommunication network recovery, performs poorly because it looks at a smaller set of routes while selecting one for the channel, and because the longer paths found by this scheme use more resources. This leads less of the affected realtime load being rerouted at high network loads. Also, as implemented in our simulation, with a complete round trip to balance resources along the path, Local rerouting did not lead to a reduction in the time to reroute. If we had devised a scheme more faithful to *link rerouting*, which does not require a complete round-trip establishment, we would definitely have gained in terms of time to reroute, but would have been able to reroute even less realtime load, because many of the channels would fail to meet their delay requirement along the longer path if we did not adjust this delay along the entire path as we did in our implementation. This seems to rule out Local as a method of constraining the route selection.

Hybrid performed almost as well as Global in terms of success of rerouting. All of the timing schemes were almost identical in terms of time to reroute, because of the round trip for establishment. But in terms of resource usage efficiency, Global was the best, and, since the performances of Global and Hybrid are similar in all other respects, Global appears to be the correct approach to use for this aspect of the design.

Adding Global prediction to a scheme without retries allows us to recover a significantly larger amount of realtime load. However, even more realtime load can be restored by adding Delayed retries, and in this case the timing of the entire system is dominated by that of the retry process. Thus, the gains of using Global prediction are diminished. In fact, since all the establishment attempts need to start at the same time for Global prediction to work well, this adds a small, but not negligible, amount to the time to reroute. In addition, the Global prediction algorithm takes more computation time to run. To extrapolate this to other topologies, we note that the synchronization and computation time for Global prediction will grow with the number of sources affected by the fault and with the size of the network; in a large network, this will be an even larger problem. Thus, we reach the conclusion that Global prediction does not seem to be useful as a mechanism to improve the reroute success.

The effect of reroute timing on the amount of realtime load rerouted was significant, but this gain came at the expense of increased time to reroute. The improvement in performance attained by adding retries exceeded what could achieved by randomized timing, with lower increase in the time to reroute. In fact, the initial ideas that we developed for this internal factor were transferred to the retry policy when we added delays to the retry scheme. In the presence of Delayed retries, Immediate timing seems to be the best approach.

Of the external factors considered, load seems to have the biggest effect on the metrics of performance. Increasing the load reduces the amount of realtime load that a recovery scheme can reroute. For the schemes using retries, the effect on the amount of realtime load can be partially offset by trading off the time to reroute. By using retries, the scheme takes more time, but manages to reroute more realtime load. However, there is a limit on the extent to which this trade-off can be exploited, since beyond a certain load level, which depends on the network's capacity and topology, the amount of realtime load that can be rerouted drops off, regardless of how long the scheme spends retrying. Also, there is a practical limit on how long a rerouting scheme should take, because the affected channels are not transmitting data during this interval, and the applications using the channels will eventually time out.

We observe that the schemes degrade gracefully with increasing load. However, in order to give satisfactory performance, the network should be run at a load such that, in the event of a failure, a large fraction of the channels can be rerouted. This is also consistent with another requirement: that, in the absence of network faults, the probability that a new request is accepted should be high. If the network is well dimensioned, both these requirements should be satisfied. The problem of network dimensioning is not dealt with in this thesis, but the maximum load at which a given network should be run, given a target percentage of affected realtime load which we would like to recover within a given time in the event of a fault, can be evaluated from our simulations.

The number of simultaneous faults being dealt with has some affect on the metrics of performance. This works in three ways: less of the network capacity and redundancy is available for rerouting; there are a larger number of channels to reroute, so the recovery schemes are stressed more; and the multiple sets of channels being rerouted are physically separated by some distance in the network, so the route update process exhibits different timing characteristics. The overall observed effect on the performance, in the case of double failures and the Delayed retry scheme, is that the recovery takes a little longer, but can still reroute as much realtime load as the Sequential timing scheme.

The last two external factors, the load mix and the topology, do not seem to have a significant effect on the recovery success of the schemes. The time to reroute, of course, depends on the topology, and would increase with the average path length of the channels to be rerouted. However, this is also true of the delay bounds that can be offered on the channels. This implies that in any network on which reasonably low delay bounds can be offered, our schemes would be able to reroute a significant portion of the affected traffic in a reasonable time. The excess resource usage did not change significantly with the topology. Thus, the schemes are relatively insensitive to the nature of the network topology. We designed the schemes to be scalable and flexible, without any target load mixes or topologies in mind. The insensitivity of the schemes to these factors validates our design choices. We must bear in mind, however, that the topologies tried in our experiments are all well-connected topologies, and that our results would not hold for poorly-connected topologies. This cannot be helped, since the basic idea on which the schemes are based is that of finding alternate paths between the source and destination; if none exists, the scheme will not work. However, our experiments seem to indicate that the results are not restricted to any specific well-connected topology.

The scheme with the best performance according to all our metrics of performance is the combination of Global locus of reroute, Immediate timing, Delayed retries, and Local prediction. The salient features of this scheme is summarized in Table 4.2. As with all presented in

Property	Evaluation
Overhead in the absence of faults	None
Amount of unrecoverable realtime load	As low as possible with greedy algorithms
Duration of disruption	Depends on topology
Guarantee of recovery	None
Overhead in the presence of faults	Small
Adaptability to external factors	Good
Complexity	Moderate

Table 4.2: Properties of the fault recovery scheme selected

this chapter, it has no overhead, in terms of resources required or channel management activity, in the absence of faults. It reroutes as much of the affected traffic as can be expected from any greedy algorithm based on the Bellman-Ford algorithm. Since the Bellman-Ford algorithm minimizes the resource usage of each individual route computation, we should no expect any better performance without discarding the greedy paradigm. The time to reroute, which determines the duration of the disruption, depends on the topology. In the topologies tested it was found to be acceptable (average 200 ms on the mesh topology). This value is related to the average round-trip time along the channel; this is because all the major components of the time to reroute, such as the time for an unsuccessful attempt, the delay before the next retry, and the successful attempt, are all related to the round-trip time. For most applications of realtime networks, the round-trip time has to be small to allow interactive communication without intolerable delays. Thus, the recovery time offered by this scheme may be acceptable on most realtime networks. However, this recovery is not guaranteed, since, under high network load, some channels may not be successfully rerouted.

In the presence of faults, the overhead of this scheme is small, since it uses Global locus of reroute, which minimizes the resource usage to the extent possible. It is also fairly adaptive to the external factors considered, such as topology, load, load mix, and number of failures. Finally, the scheme is among the simplest of the schemes that we considered, from the point of view of implementation. This point is considered in more detail in the next chapter, which deals with the high-level design of a protocol embodying the ideas presented here and in the last chapter.