Recovering Guaranteed Performance Service Connections from Single and Multiple Faults†

Anindo Banerjea, Colin J. Parris, and Domenico Ferrari.

Tenet Group
Computer Science Division, UC Berkeley
and
International Computer Science Institute,
1947 Center St. , Suite 600
Berkeley, CA 94704-1105.
Tel: (510)-642-8919 Fax:(510)-643-7684
E-mail: {banerjea,parris}@tenet.berkeley.edu

TR-93-066

ABSTRACT

Fault recovery techniques must be reexamined in the light of the new guaranteed performance services that networks will support. We investigate the rerouting of guaranteed performance service connections on the occurrence of link faults, focussing on the aspects of route selection and establishment in the network. In a previous investigation, we explored some components of rerouting in the presence of single link faults in the network. In this paper we study the behavior of our techniques in the presence of multiple link faults in the network, and also examine the technique of *retries* to improve the success of rerouting. Our schemes are simulated on a cross-section of network workloads, and compared using the criteria of the fraction of the affected traffic that could be rerouted, the time to reroute and the amount of resources consumed in the network. A novel metric, the *Queueing Delay Load Index*, which captures both the bandwidth and delay demands made on the network by a connection, is used to present and analyze the results.

Keywords: Fault recovery, guaranteed performance, network management.

†This research was supported by the National Science Foundation and the Defense Advanced Research Projects Agency (DARPA) under Cooperative Agreement NCR-8919038 with the Corporation for National Research Initiatives, by AT&T Bell Laboratories, Hitachi, Ltd., Hitachi America, Ltd., Pacific Bell, the University of California under a MICRO grant, and the International Computer Science Institute. The views and conclusions contained in this document are those of the authors, and should not be interpreted as representing official policies, either expressed or implied, of the U.S. Government or any of the sponsoring organizations.

1. Introduction

Guaranteed Performance Communication[1] (GPC) has been recognized as an important service from future wide-area broadband-ISDNs [23]. Several GPC schemes have been proposed to date; however, they all assume adequate fault-free service from the underlying network. The high level management features needed for such schemes are still under investigation. Fault recovery is one of the critical features which must be added to make GPC schemes of practical use.

Fault recovery consists of the tasks of fault detection, recovery instigation, computing and distributing new state information, and rerouting the affected traffic. Our investigation focusses on the *rerouting* of GPC connections in a network with redundant connectivity after the occurrence of link faults. In a previous paper [24] we investigated rerouting to recover from single faults in the network along two orthogonal components. In this paper we extend the investigation to multiple faults, which aggravates the problem of *route collisions* (see Section 4.2). We also borrow the idea of retries from the world of telecommunication failure recovery [10], and explore the effect of adding retries to our schemes under single and multiple faults.

In Section 2 we look at related work in the area of GPC schemes and network fault management. In Section 3 we examine the reasons for choosing this approach to fault recovery. In Section 4 we present a model of the network in which our fault recovery schemes work, and a model of the fault recovery process. In Section 5 we present our experimental design. In Section 6 we present and analyze our results. In Section 7 we summarize our results and mention directions of future research.

2. Related work

In [24] we investigated fault recovery for GPC networks in the presence of single link faults along two components: the *locus of reroute* (which determines the section of the failed route over which a reroute is attempted) and the *reroute timing* (which determines when the actual attempt to acquire resources over the new route is performed). These approaches and the results of our investigation are summarized in Sections 4 and 6. Apart from this paper, we have not come across any investigation of fault recovery for GPC schemes in the literature. However, a significant amount of work has been published in the area of GPC schemes (without fault recovery), and of fault recovery in other contexts.

Several investigators have proposed schemes for GPC service in packet switched networks. The Tenet group [2] has devised an approach [1] to the provision of such services, which we assume as part of our underlying network model and describe briefly in Section 4. The *guaranteed service* proposed in [2] meets the definition of GPC as used in this paper, as opposed to the *predictive service* proposed in the same

[1] Defined as communication with guaranteed bounds on performance parameters such as bandwidth, delay, jitter and loss rates.

[2] At the University of California at Berkeley and the International Computer Science Institute.

paper. The Session Reservation Protocol [3] provides deterministic guarantees and follows an approach similar to that of the Tenet group. The Heidelberg Resource Administration Technique (HeiRAT) [5] uses the ST-II protocol to reserve bandwidth resources for unicast and multicast connections. The Asynchronous Time Sharing (ATS) approach [4] provides a menu of QOS classes to the client, while the Flow Protocol [25] allows clients to specify an average throughput bound and compute the resulting delay bounds. PlaNET [26] performs bandwidth reservation using an equivalent bandwidth model. All these schemes have some basic similarity; indeed it can be argued [1] that any scheme that provides a similar service with the same set of assumptions about the network must to have the properties of resource reservation, admission control, connection-orientedness with pre-computed routes, and a service discipline or policing strategy which protects one connection from another. This leads us to believe that it may be possible to devise an abstract model of GPC schemes which is independent of the details of the particular scheduling mechanism, admission control test, etc. but captures enough information to allow management schemes to be built on top of these data transfer schemes. Our fault recovery schemes make as few assumptions as possible about the underlying GPC schemes, hence we believe that they would work well with any of the above GPC service schemes.[3]

Fault recovery in telecommunication networks has been presented in [6] in the context of a layered model of the transport network. At the highest layer, the *switched layer*, the unit of communication and fault recovery is the *call*. The recovery action taken at this layer is to update the routing database in order to correctly route new calls around the fault [7,8]. Existing calls are lost and have to be redialed by the end systems. At the *Cross-connect layer* the units of communication and recovery are *trunks* (e.g. DS1 or DS3),[4] which have fixed bandwidth requirements and relatively stable routes. This makes the network relatively stateful and the fault recovery mechanism attempts to preserve this state. Recovery approaches used at this level include pre-computing and storing the configurations for a large failure set [9,10,11], and running dynamic distributed algorithms to find short routes [12,13,14,15].

Conventional computer data networks are mostly based on a connection-less paradigm (e.g. Internet). Fault recovery in these networks consists of recomputing routing information to route new data correctly. In the Internet no performance guarantees are given, though the protocols attempt to reduce congestion and network instabilities [16,17,18]. AN1 [19] and AN2 are local area networks designed for high survivability. AN1 is packet switched and connection-less, while AN2 is ATM based, connection-oriented and supports CBO (fixed bandwidth) traffic. However, both networks stop data forwarding during the network state acquisition and routing table computation; this technique only works in local area networks.

---

[3] However our simulations test the performance of these schemes in the concrete context of the Tenet scheme, since the resource reservation and scheduling disciplines are simulated in detail.

[4] The trunks in turn appear as the links in the network topology seen by the next higher layer: the switched layer.

3. Motivation

Fault recovery for GPC networks is different from that for conventional data networks because GPC networks need to maintain performance bounds and keep network state during recovery. It has more in common with recovery in the cross-connect layer of the telecommunication network, where the trunks have bandwidth requirements and relatively stable paths. However in the cross-connect layer, there are relatively few trunks which belong to a small number of classes (e.g. DS1, DS3), whereas in a GPC network the connections would in general have peak and average bandwidth specifications, delay, jitter and loss-rate requirements, and the number of possible combinations preclude approaches such as pre-computation and storage of all configurations. As such, more distributed dynamic schemes, which provide good but sub-optimal solutions, have to be investigated.

Our investigation focusses on the aspect of rerouting, where we think the hardest problems of failure recovery lie. One possible approach to fault recovery is to move this function to the application layer, i.e. recompute the network topology and state so as to route new requests correctly, but allow all affected connections to be lost and re-established by the applications. This idea is clean and appealing, but there are strong practical reasons to incorporate at least some fault recovery within the network. Application initiated fault recovery would result in all the affected connections being retried within a short time after the fault, especially if the applications have requirement for quick recovery. Thus all affected connections will compete for resources during the rerouting phase, causing *route collisions*. Spreading the attempts out in time and space (over different routes) would increase the probability of success of the attempts. This sort of cooperation is harder at the application level. In addition, there is state within the network which can be reused to reduce the amount of work which needs to be done. It may be possible to restrict the exchange of messages to the affected portion of the connection to speed up the recovery and increase the probability of success. The response time of the reroute will also be faster, the lower the level at which this recovery action is initiated. For instance, if the recovery action is initiated by the node adjacent to the failed link, then the recovery will be faster than if the error message must first propagate to the application layer. Another (perhaps debatable) advantage is that the user is not involved in the recovery action. Finally, a scheme for automatic recovery could also be used to move traffic off a link, in response to increased error rates or as a management action.[5] In this case, only the fault detection mechanism would need to be modified to allow the recovery mechanism to be triggered manually or by error rate thresholds. Thus there are strong reasons to consider placing recovery mechanisms within the network. It is therefore important to explore the various possibilities for network initiated automatic rerouting systematically.

Our previous investigation explored two components of rerouting against a cross-section of network loads using evaluation criteria that captured the impact of the fault on the client and the network. In this paper we add the component of multiple retries, borrowing the idea from the telecommunication world

[5] For example, to allow a connection with unusually high requirements to be established, or for maintenance.

[10], and compare the possible benefits resulting from this approach against the other techniques. We also examine in this paper the effect of multiple isolated faults within the same *recovery cycle*,[6] where isolated means that the faults affect disjoint sets of channels. Since the logical topology seen by computer networks is actually overlaid on a lower physical topology, with logically separate trunks sometimes existing on the same physical fiber, multiple isolated faults are not such an unlikely occurrence. Multiple non-isolated faults are more unlikely since they would arise from the simultaneous failure of two distinct physical links.[7] We would like to look at the effect of the temporary increased routing load caused by two simultaneous faults trying to use the remainder of the network to reroute their affected channels. Our earlier investigation shows us that the major difficulty with rerouting in GPC networks is route collisions and we expect multiple isolated faults to aggravate this problem.

4. Model

In this section we provide the network model. This model includes the model of the guaranteed performance scheme (i.e. the Tenet model) and that of the fault recovery scheme. In conjunction with the detailed description of the fault recovery model, the underlying support mechanisms (i.e. the fault detection and control mechanism and the routing mechanism) will also be presented.

4.1. The Tenet Scheme

The Tenet scheme allows the network to provide guaranteed performance communication services. In this service a guaranteed performance connection (*a real-time channel*) is an abstraction that defines communication services with guaranteed traffic and performance parameters in a packet-switched network[1]. A channel's traffic is characterized by: $X_{\min}$, the minimum packet inter-arrival time, X_{ave}, the average packet inter-arrival time over an averaging interval I, and, $S_{\max}$, the maximum packet size. The performance requirements available to channel are: D,the maximum delay permissible from the source to the destination, J, the maximum delay jitter[8], Z, the probability that the delay of the packet is smaller than the delay bound, and, W, the buffer overflow probability. In the Tenet scheme a channel is *established* and then data is transferred. Channel establishment occurs in the following sequence: a real-time client specifies its traffic characteristics and performance requirements to the network; the network determines the most suitable route for this channel using its traffic characteristics and performance requirements; the network translates the end-to-end parameters into local parameters at each node, and attempts to reserves resources at these nodes accordingly. Channel establishment is source routed and occurs in two passes (i.e.

[6] The interval from the occurrence of a fault to when the last connection affected by that fault is either successfully rerouted or reroute attempts are aborted.

[7] We assume that the logical network layout and the routing on top of that does not cause a connection to traverse the same physical link twice.

[8] Jitter is defined here as the difference between the delays experienced by any two packets on the same connection.

the round trip from the source host to the destination host). Resources, corresponding to the lowest possible delay, are reserved at each node along the forward pass. This is required because the exact reservation required to meet the end-to-end requirements depends upon the resource state on the downstream nodes; thus, a scheme which reserves less resources at an earlier node might fail to meet the end-to-end delay requirement, if a downstream node is heavily loaded and cannot provide a low delay bound. Our scheme thus over-books resources on the forward pass, and along the reverse pass these resources are appropriately relaxed to reflect the actual needs of the channel. To determine the availability of resources, each node uses the Tenet algorithms or admissions tests, which are based on the service discipline. Research by Ferrari et al.[20] has shown that a broad spectrum of service disciplines may be used in the Tenet framework, however, in this paper we have chosen Rate Controlled Static Priority (RCSP) queuing as it decouples the bandwidth and delay resources while providing simple admissions tests.

4.1.1. Rate Controlled Static Priority (RCSP) Queuing

The RCSP service discipline can be logically divided into a rate controller and a scheduler. The rate controller ensures that all sources obey their input traffic description by calculating the expected arrival time of a packet using its traffic description. If at any time a packet violates its traffic description by arriving before its expected arrival time, this packet is held until such a time that its traffic description is not violated. Packets are then passed on to the scheduler which maintains several levels or priorities and serves packets First-Come-First-Serve (FCFS) within each priority starting at the highest priority queue at which packets are waiting and ending at the Non-Real-Time queue[9]. The separation of the rate controller and the scheduler provides the decoupling of the bandwidth and delay resources.

The admission control test, provided below, is performed at each node during channel establishment in order to provide guaranteed performance service communication in a network of servers with RCSP scheduling. The test proceeds as follows: For a request with the traffic specification ($X_{min}, X_{ave}, I, S_{\max}$) and a delay bound requirement D_k (where $D_1, D_2, D_3, .., D_l$ are the delays associated with each of the l priority levels in the switch), if for all priority levels p greater than or equal to k:

$$\sum_{j=1}^{j=c_p} \left\lceil \frac{D_p}{X_{\min_j}} \right\rceil * S_{\max_j} + \left\lceil \frac{D_p}{X_{\min}} \right\rceil * S_{\max} + S_{maxP} \le D_p * L \qquad (1)$$

then the new channel can be accepted (at the priority level k). c_p is the current number of channels at or above the priority level p, $X_{\min_j}$ is the minimum interarrival time of packets corresponding to the jth connection at that priority level, S_{maxP} is the largest packet size that can be transmitted over the link, and L is the link speed. Proofs of the correctness and sufficiency of this test are presented in [22].

The purpose of the test is to guarantee that a packet belonging to any priority level p experiences at most a delay bound of D_p before it is transmitted, assuming that all previously queued packets at this level

[9] Packets in the Non-Real-Time queue are only served after the lowest level RCSP queue's packets have been served.

and all packets at higher priority levels will be transmitted prior to it. Therefore, adding a channel at level k contributes *work* to all equal and lower priority levels $p \geq k$. The admission test ensures that the total *work* at any level p cannot exceed the delay bound D_p for that level, where *work* at level p is defined as

$$W_p = \sum_{j=1}^{j=c_p} \left\lceil \frac{D_p}{X_{\min_j}} \right\rceil * \frac{S_{\max_j}}{L} + \frac{S_{maxP}}{L} \qquad (2)$$

4.2. The Failure Recovery Model.

In this section we will present our model of failure recovery which is based on the Tenet guaranteed performance services scheme. We will begin by giving a brief overview of rerouting to provide the proper context for our work. We will then describe our model and discuss the specific components that we intend to investigate.

As mentioned previously, fault recovery can be divided into the tasks of fault detection, recovery instigation, computation and distribution of new routing information, and channel rerouting. The focus of this investigation is the rerouting task, which can be further subdivided into *route selection*, determining the new route, and *alternate route establishment*, transferring the channel onto the new route. Optimal solutions of *route selection* involve exponential computational complexity, and all other solutions involve trade-offs between time and goodness of the solution. *Alternate route establishment*, which changes the resource state in the network to reflect these new routes, involves cooperation, since network resources must be shared.

In the literature [6], the approaches to rerouting have been classified along three axes: *Centralized vs. Distributed schemes*, *Link rerouting vs End-To-End rerouting*, and *Pre-computation vs dynamic computation of routes*. In an integrated high-speed packet-switched network that provides guaranteed performance service connections, we believe the most useful approach to be a distributed, dynamic computation approach. The problems introduced by a centralized scheme, i.e. scalability, reliability of the central control unit, and congestion and computational bottlenecks at the control unit, lead us to believe that a distributed scheme would be more efficient. Also, the dynamic nature of multimedia traffic, which will most likely pervade these networks, will render the pre-computation of routes ineffective. Rerouting will need to be done dynamically. In our investigation we will compare Link and End-To-End rerouting.

In our failure recovery model, a fault is detected by the node upstream of the faulty link, which informs all nodes in the network of the topology change by route update messages. The detector also instigates recovery on affected nodes (i.e. source nodes of affected connections) by sending a reroute message. These nodes then perform route computations and attempt to reroute as many of the affected connections as possible, while satisfying their initial traffic and performance requirements. Our model for fault recovery contains the following components: the method used for fault detection, the routing algorithm used to find a new route for a given channel, the constraints on the route selection, the timing of the alternate establishment attempt, and the maximum number retries that are performed. In our investigation, we

focus on the last three components, which together comprise the rerouting scheme. We fix or make simplifying assumptions about the other components.

We examine rerouting along the lines of three orthogonal components: *the locus of reroute*, *reroute timing*, and *retries*. One of the goals of this investigation is to establish the significance of these components, as such we have chosen easily analyzed yet broadly encompassing instances of each. The *locus of reroute* component indicates the node which performs reroute selection, and the portion of the current route traversed by the connection upon which a reroute attempt will be made. We consider three locus of reroute types: *Global reroutes*, *Local reroutes*, and *Hybrid reroutes*. With a Global reroute, which corresponds to the *end-to-end* rerouting of [6], the path of the rerouted connection is determined at the source node based on the current network load and the traffic and performance characteristics of the connection. As all information available to the node is considered in determining this route, this scheme would tend to distribute load evenly through out the network. With a Local reroute, the rerouted connection traverses the original route but bypasses the failed link. The route computation is performed by the node upstream of the failed link, and determines a path segment from this node to the node downstream of the failed link. This segment is combined with the unaffected portion of the old route, removing any redundant links or loops. The *route establishment* still traverses the entire new route, in order to balance resource along the length of the route. This type of locus of reroute reuses the maximum amount of previously reserved resources, and in a highly loaded network may have a higher probability of success than a Global reroute. It also localizes the effect of the reroute to the immediate neighborhood of the fault, which might improve performance for multiple faults. It corresponds roughly to *link rerouting* of [6], except that the complete round-trip for *route establishment* implies better resource balancing, higher success rates, but longer establishment times. A Hybrid reroute combines the previous two types of locus of reroutes; a Local reroute is first attempted, however, if a local path cannot be found, a Global reroute is initiated. In all types of locus of reroutes the traffic and performance characteristics of any rerouted connection must be maintained after the reroute.

The *reroute timing* component provides the starting time of the reroute attempt. This component seeks to mitigate the effect of route collisions and database inconsistencies. *Route collisions* occur when an establishment request, during the forward pass along a link, consumes a large fraction of the remaining resources on the link (see Section 4.1), thus causing an immediately following forward pass establishment request to fail due to lack of resources, even though on the reverse pass the former establishment request would have relaxed its resource reservations such that the latter could have had its resource request honored. Reducing route collisions would increase the probability of establishing a connection. *Database inconsistencies* can also cause reroute attempts within a small time interval of each other to interfere. Had there been sufficient time between connection reroute attempts the routing database may have reflected a more accurate network load and the latter of the two reroutes would have chosen a different path through the network.

Three possible approaches are considered for the timing component: *Immediate*, *Random(n)*, and *Sequential*. Under Immediate timing, reroute attempts are initiated by the controlling node as soon as a failure is reported to it. This approach depicts the least cooperation and possibly the most expected interference. Under Random(n) timing, the reroute attempt time is determined by generating a random value from a uniform distribution over an interval of duration n. This random value is added to the current time with the result being the reroute starting time. This approach does introduce some level of cooperation as all controlling nodes use the same algorithm. Under Sequential timing, all reroutes attempts are handled sequentially with only one controlling node initiating a reroute attempt at any given time. When all the reroute attempts of a single affected connection are completed only then are reroute attempts made on another connection. Sequential timing reduces route collisions and database inconsistencies but at the cost of much higher average time to reroute.

The *retries* component indicates if retries are permitted on a reroute attempt. The retry mechanism removes the saturated link (at which the previous reroute failed) from consideration, incorporates new information presented by recent database update messages, and attempts to calculate a route from the source to the destination. Retries are repeated on failure up to a pre-determined limit; different values of the limit give us different possible approaches within this component.

The recovery schemes presented (i.e. all combinations of the approaches of the three components) are fairly simple and may not translate directly to practical implementations. However, as the objective of our investigation is to look at the significance and impact of the components, we choose approaches that span the entire spectrum of the each component. For example, Sequential timing with retries, while it may not be practical from the stand point of the time to reroute, will give us an idea of the best possible performance, in terms of rerouting success, which may be achieved by a perfect timing strategy with the maximum number of retries. Sequential provides a standard against which we can compare other more practical timing schemes. In our experiments we will explore the solution space of our recovery model by examining all combinations of the three components.

The rerouting schemes presented above utilize two mechanisms, the *fault detection and control mechanism* and the *routing mechanism*. The fault detection and control mechanism recognizes faults, informs the routing mechanism that new route update messages are needed, and disseminates fault messages to the source nodes of the affected connections. The process of fault detection is complex and is not investigated in this paper. We assume that when a link is characterized as faulty, low level validation has been performed to eliminate transient faults and oscillations. The fault message provides the relevant reroute scheme information to all affected connections. The routing mechanism is a combination of the routing algorithm and the route update mechanisms and is described in the next section.

4.3. The Routing Mechanism.

The routing algorithm used in rerouting is based on the DCM Routing Algorithm [21] and determines a route from the source node to the destination node taking into consideration the traffic and performance characteristics of the connection, the current network load, and the locus of control. The goals of the routing algorithm are to maximize throughput, balance the network load, obtain routes in a timely manner, and to maximize the probability that the route provided by the algorithm will be successfully established[10]. The routing algorithm calculates a minimal-cost route where the cost of the route is the sum of the costs of the links comprising the route. The cost of a link is a delay value, which is the sum of the minimum queuing delay offered by the starting node to a real-time channel with these traffic characteristics, the transmission delay, and the propagation delay along the output link. The transmission and propagation delay are fixed costs, however, the queuing delay in the scheduler is variable, and dependent on the current channel resource reservations and the traffic characteristics of this new channel (see Section 4.1.1., equation 1).

The DCM routing algorithm proceeds in two steps. In the first step a directed graph is created in which the nodes correspond to switches and hosts in the network and the edges to the links connecting these switches and hosts. The weights attributed to each edge represent the link costs. The link cost are computed just prior to applying the algorithm thereby using the most recent link information obtained from routing update messages. In the second step a constrained, modified version of the Bellman-Ford[11] algorithm is then applied to this graph to determine a possible route. In this algorithm consecutive searches are performed on all 1, 2, .., N-2-hop paths from the source to the destination node until the delay condition $\sum_{l(s,d)} d_l \leq D$ is satisfied, where D is the delay bound of the channel, d_l is the weight of link l, (s, d) is the path from the source s to the destination d, and N is the number of nodes in the network. A *constraint* is placed on the number of possible searches by stopping at the first *hoplevel*[12] at which the delay bound condition is satisfied, and choosing the minimum cost path within the same level.

The DCM routing algorithm maximizes throughput by minimizing the number of intermediate nodes encountered along the path from the source to the destination host. The load balancing criterion reduces call blocking by distributing the load more evenly throughout the network. The algorithm limits its search space, thus reducing its computation time, thereby providing routes in a timely manner. It also increases the probability that channel establishment will be successful as it determines the link queuing delays based on the traffic characteristics of the channel and the most recent resource reservation information.

Routing updates are currently done on a per-channel-establishment basis. This is accomplished by having every node broadcast the load values of its links to all other nodes after it sends the reverse channel

---

[10] That is, the route will be established with the traffic and performance specifications given by the client.

[11] The fundamental Bellman-Ford algorithm [22] searches for the shortest paths between a specified source and destination node starting from all possible one-hop paths and continuing until the N-2-hop paths are examined.

[12] This *hoplevel* is the number of hops from the source to the destination node.

establishment message to the previous node on the new channels route; each receiving node updates its local link-state table. This broadcast is done along a minimum spanning tree.

5. Experimental Design

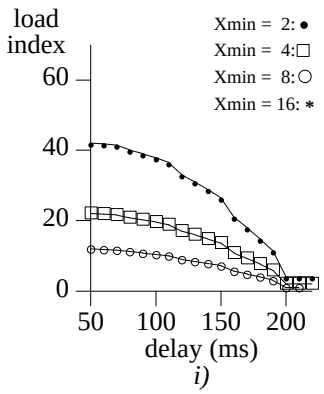
This section provides a framework for our experiments by discussing the metrics by which we evaluate the various rerouting schemes, and the other experimental factors that we need to consider. First we present a new index, the *Queuing Delay Load Index* that will be used to characterize the network load and to define our performance metrics.

5.1. Load Index

A major problem to be addressed by any investigation which seeks to compare schemes involving guaranteed performance connections is the choosing of an index which is able to characterize the network load imposed by one or more guaranteed performance connections. As this imposed load is dependent on the number of connections, their bandwidth, delay, and jitter requirements, and the links traversed by each of these connections, a general solution to this problem is quite complex. However, we have devised a simple index which meets the above requirements using the concept of *work* as defined in the RCSP (Rate Controlled Static Priority) service discipline.

In RCSP, placing a channel in the network at priority level l adds work to all lower levels. This is evidenced by a low delay channel adding a greater amount of work, since it is placed at a high-priority level (small l) thereby increasing the call blocking probability at all levels $m > l$. If the delay requirement is larger, it can be placed at a lower priority level (larger l) effecting fewer levels and contributing less work. This situation prompts us to consider the sum of the *work* (specified by equation 2) across all levels as an index of the load on the node. A large increase in this index will be caused by a high priority channel as it will reflect the *work* component at a greater number of levels, while a channel at the lowest priority level will only reflect the work done at one level. We call this index of the load on the network the Queuing Delay Load Index.

Figure 1 depicts the ability of this index to capture the effect of the bandwidth, delay and path-length components in the overall load on the network. Graph *i* illustrates the response of the index with respect to the delay requirement of a channel on an empty network. As shown in the graph the load index decreases as the delay requirement is lessened. The graph flattens out beyond 200ms as the connection is now placed at the lowest level in the RCSP queues and further increases in the delay bound do not effect the priority level of the channel. Therefore with all other components fixed, the index decreases monotonically with respect to the end-to-end delay bound. Graph *ii* indicates that the load index increases linearly with respect to the bandwidth of the traffic, with the gradient of the line dependent on the delay bound of the channel. Graph *iii* shows the linear increase of the index with respect to the path length (the delay bound and the bandwidth are constant). Finally, graph *iv* shows us that the index is additive, i.e. two channels existing


load
index
Xmin = 2: ●
Xmin = 4:□
Xmin = 8:○
Xmin = 16: ∗
60
40
20
0
50
100
150
200
delay (ms)

i)