ZRESTORE: A LINK RESTORATION SCHEME WITH HIGH AGGREGATION AND NO RESERVATION

Shirshanka Das[1], Pavan Verma[2], Bijendra Nath Jain[3], Mario Gerla[1]
[1] *University of California, Los Angeles*
{shanky,gerla}@cs.ucla.edu
[2] *University of Michigan, Ann Arbor*
pverma@eecs.umich.edu
[3] *Indian Institute of Technology, Delhi*
bnj@cse.iitd.ernet.in

Abstract This paper proposes a new link restoration scheme for optical networks. Our scheme uses dynamic provisioning of bypass routes to effectively utilize bandwidth for restoration. The bypasses are pre-selected but not reserved. Bandwidth is requested in aggregate, resulting in control traffic savings. A unique idea of over-protecting channels on the links for future work savings has been proposed in this paper. Our scheme assumes the knowledge of global topology (but not capacity) information at each node. Using simulation results we show that our scheme is very competitive in terms of restoration times and restorability even in very high load conditions.

Keywords: Optical WDM networks, restoration, dynamic, distributed.

1. Introduction

With SONET being pushed out of optical networks, and demands for the same degree of reliability, a lot of effort is being spent in analyzing optical WDM networks with regard to finding ways to protect them against failures.

A restoration scheme is usually evaluated by four important performance criteria: *restoration speed, restorability, capacity efficiency and algorithm scalability*. Out of these four performance evaluation criteria, *restoration speed* is the key concern for optical networks, but *capacity efficiency* is also gaining importance, since that is where mesh has more promise over ring architectures.

In this paper we propose a novel dynamic way of looking at the link restoration problem. Most of the pre-computed schemes that exist today are static in the sense that they compute bypass routes once and reserve it for restoration.

A. Bianco et al. (eds.), *Next Generation Optical Network Design and Modelling*

Doing so leads to bandwidth wastage as demand patterns in a network may vary greatly with time and thus we might end up trying to protect many more channels than actually required.

This paper is structured as follows. In section 2 we discuss previous work in the areas of link and path restoration in optical networks. In section 3 we give an overview of ZRESTORE to familiarize the reader with the basic ideas. In section 4 we give the details of ZRESTORE at the level of control packets and datastructures maintained at the nodes. In section 5 we describe the restoration procedure, in section 6 and 7 we discuss the simulation process and the results. We talk about certain extensions to the scheme in section 8 and end with our conclusions and possible future work in section 9.

2. Previous Work

Several distributed precomputed approaches have been proposed in the past. In these strategies, restoration paths are failure dependent, which may allow better capacity utilization. However, a large number of restoration paths - one set for each failure scenario - must be pre-calculated and stored in memory. This creates a scalability issue. Moreover, potential difficulty in rapid failure isolation in optical networks makes this approach unattractive.

Schemes which use disjoint link-node paths for the restoration and the primary routes are generally capacity efficient since they promise 1:N sharing of resources. However, practically the degree of sharing that really happens depends a lot on how the primary paths were setup in the first place. Therefore, for such schemes to succeed, there needs to be a correlation between the method used to select the primary route and the restoration route. This joint-optimization problem is NP-complete and solutions are typically pretty complex[1, 3]. Besides, such path-based schemes involve signalling the whole path after the failure has occurred resulting in longer restoration times than if a much shorter bypass for the failed link or node had been setup. Another drawback of such schemes is the implicit assumption that the whole network is biconnected. In practical situations, there may well be cases such that there does not exist an alternate node-link disjoint path for a particular path due to a critical link. In such situations the "best-effort" restoration, that is, restoration on as many links as possible that is provided by link-based schemes becomes more attractive.

Distributed methods may involve precomputed route tables or real-time discovery of capacity and routes. Real time capacity discovery based methods send out flooding messages that search for available link capacities after a link failure happens and then reroute the disrupted traffic around the failed link[2]. The advantages of these link-based, real-time techniques are their simplicity and distributed nature. The disadvantages are that typical link-based schemes

are saddled with are their slowness (due to the real time nature), inefficient capacity utilization(due to sub-optimal bypass selection), and restorability limited to single-link failures (handling node failures is non-trivial in typical link-based schemes). Thus, a link-based scheme which can remove these disadvantages is attractive for the problem at hand.

3. ZRESTORE: An Overview

In this section, we first give a step-by-step description of the main features of ZRESTORE and after that a brief overview of the protocol to familiarize the reader before we get to the actual details in section 4.

The architecture that we assume is a WDM optical network with crossconnects capable of wavelength conversion at every node. We also assume a dedicated control channel for inter node signaling. ZRESTORE is a link restoration scheme based on pre-computing bypass routes. Bypasses for a link are precomputed and remembered to be used for rerouting traffic upon failure of the link. In ZRESTORE bypass routes are only *remembered* and not reserved, thus primary traffic is never blocked due to backed up channels.

ZRESTORE uses channel aggregation while searching for the bypass routes. This means that while doing the search, we don't search for just one lightpath, but for a collection of lightpaths thus protecting chunks of channels at once instead of doing it incrementally. This lends to greater scalability with number of channels per link. However at the same time, we do not sacrifice any flexibility in bypass route selection for each individual channel.

A key feature of our algorithm is that the information stored at all nodes is only the number of channels and not the identity of the channels themselves. This leads to advantages in storage and aggregation and fault-time decision of which channels to restore and which not to.

The main requirement for ZRESTORE is that each node be aware of the global network topology for computing the potential bypass routes. As link bypasses tend to be short, a clever node may only be aware of topology information in its neighbourhood and still effectively use ZRESTORE.

At birth (either at the inception of the network, or when a link is introduced in the network, or when it comes up after a failure), each link tries to get backup bypasses for half the total channels on the fiber (at this point the number of active channels is zero). As connections get set up and backup paths start getting kicked out, each link waits until the number of primary paths through it gets really "close" to the number of restorable channels. This decision of "closeness" depends on a threshold T that can be set. Now, when the threshold is reached, the link will again initiate a bypass search procedure in the background, not for just one channel, but for a target η no. of channels. The idea is to protect more than the bare minimum. However, we would not like to protect too many

channels as that would mean greater control traffic overhead when primary routes are setup on links on the bypass. Therefore a good choice of the target η is very important. The source end of a link is responsible for conducting bypass searches, whereas information about bypasses is stored at the destination end. The restoration process is therefore destination node driven. This is favorable because typically in optical networks, upon a link failure the destination node gets to know about the failure earlier than the source node. The reservation of the free channels for restoration is done only after the fault occurs, therefore no bandwidth is blocked for path setups.

In a scheme like ZRESTORE, reservation of channels would lead to a lot of bandwidth wastage. As connections in a network are set up and torn down the number of active channels on a link will vary greatly. Thus the restoration requirements would also vary, and keeping consistently accurate backed up bypasses would be close to impossible and also involve a lot of extra signaling.

Thus we intend to derive the maximum benefit of aggregation, pre-computation and non-reservation through this scheme.

4. ZRESTORE: Protocol Description

4.1. Information: requirement and storage

ZRESTORE partitions the total bandwidth (λ_{max}) of each link into the following subsets.

- α = number of channels being used for primary routes (active channels)
- β = number of spare channels, i.e. $(\lambda_{max} - \alpha)$
- δ = number of restorable channels ; a restorable channel is one for which a bypass has been found and informed
- γ = max(number of channels promised for restoration to another link), i.e. $\gamma = \max(N_L), L \in$ *promised channels list.*

Some other key terms that are used are:

- *promised channels list*: a list of links using my channels for restoration. Entries in the list are of the form (L, N_L) where N_L is the number of channels link L has been promised.
- θ = no. of simultaneous alternate bypass route searches that are done in a single pass during the route search procedure. We do not expect that a single bypass will be able to completely fulfil the request. Therefore we do a parallel search over θ prospective bypass routes.

- η = no. of aggregated channels for which the search procedure is initiated. It is an intelligently calculated "favorable target" for the number of restorable channels.
- T = threshold for the difference $\delta - \alpha$
- *critical state*: a state in which $\delta - \alpha < T$ on a link. This state causes the source node of the link to try to restore at least $T - (\delta - \alpha)$ more channels.

The information required and maintained by a link is divided into its two ends (sender and chooser) in the following way.

A sender node maintains the following for each outgoing link incident to it:

- $\alpha, \beta, \gamma, \delta, \eta, \theta$
- the *promised channels list*
- a list of bottleneck links

A chooser node maintains the following for each incoming link incident to it:

- the restoration table: a restoration entry is a tuple of number of channels and bypass, <num_channels, bypass> meaning that the bypass has promised the link num_channels channels for restoration.
- θ: the number of route-capacity-checker packets that the destination expects to receive from the source
- buffer of ROUTE-CAPACITY-CHECKER packets that have been received: this is needed to store the packets till all ROUTE-CAPACITY-CHECKER packets are received or a timeout occurs.

4.2. Bypass Route Search

The source node of every *critical* link tries to pre-compute the bypass for η channels on the link to come out of criticality. To be very precise η is the desired increase in the number of restorable channels. So the target for the number of restorable channels is $\delta + \eta$.

The method for deciding the value of η that we follow is that η is set to the value got by minimizing the sum $(\delta + \eta)$ subject to the constraints :

$\delta + \eta = \lambda_{max} * \left(1 - \frac{1}{2^i}\right), where\,(i \,\epsilon\, I^+)$

$\delta + \eta > \alpha + T$

For example, with $\lambda_{max} = 64, \alpha = 11, T = 2, \delta = 4$, η comes out to be 12.

The method used for calculating η represents a logarithmic increase in the desired number of restorable channels. What we try to accomplish here is to use the benefit of aggregation to help us in working less in the future, so that

the amortized cost of this procedure is less. However any other scheme for calculating η can be used. Setting η appropriately for restoration route searches ensures that we try to protect channels for the future only if the bypasses are relatively empty. However, we need to make sure that we don't over do it; that is we don't protect too many channels. Over-protecting may lead to a surfeit of control packets later, since primary routes will start displacing these restoration routes from the bypass links and trigger decrement control packets.

Next, the source node computes θ link-disjoint paths that can serve as a bypass for this link using its knowledge of global topology. We compute min hop alternate bypasses for the link using Dijkstra's algorithm. After this it sends out θ ROUTE-CAPACITY-CHECKER packets for checking out availability of the required number of channels on the θ bypass paths.
The contents of the ROUTE-CAPACITY-CHECKER packet are the *packet_id* (this is same for all θ control packets), the *link_id*, the *bypass_route*, *num_required* (the number of required channels = η), *minimum_free* (the minimum number of spare channels the packet gets along the bypass route) , and the *bottleneck link*, (the link which had *minimum_free* spare channels).

At the same time, the source also sends a CONTROL-INFO packet to the destination through the link itself. This packet contains information that is vital for the destination to know how long to wait. The contents of the CONTROL-INFO packet are the *packet_id*, θ , and the *timeout value* (the expected max time for all the ROUTE-CAPACITY-CHECKER packets to reach the destination). The destination of a link on receiving the CONTROL-INFO packet sets itself to start processing the control packets either after receiving θ ROUTE-CAPACITY-CHECKER packets with the same *packet id* as the CONTROL-INFO packet or after the *timeout* written on the CONTROL-INFO packet.

When u, the source of a link (u, v) receives a ROUTE-CAPACITY-CHECKER packet, it looks at the field *num_required* and modifies the *minimum_free* field on the outgoing packet if $\beta_{(u,v)} < N_{link\_id} + minimum\_free$. The new value of *minimum_free* is set as $\beta_{(u,v)} - (N_{link\_id} + minimum\_free)$. If link *link_id* is not in the *promised channels list* for link (u,v) then of course N_{link_id} is taken as 0. When the destination receives a ROUTE-CAPACITY-CHECKER packet, it updates the database of θ bypasses that it has in its memory to incorporate the information carried in it.

The destination of a link can't start processing a ROUTE-CAPACITY-CHECKER packet as soon as it is received because all the ROUTE-CAPACITY-CHECKER packets need to be processed together to correctly assign number of restorable channels to each bypass. When the last ROUTE-CAPACITY-CHECKER packet has arrived or if timeout occurs, the destination partitions the required lambdas into the θ bypasses (or the bypasses for which it received packets) using the following strategy. Keep the ratio $\frac{channels\ to\ be\ restored\ through\ bypass}{minimum\ free\ on\ bypass}$

approximately the same for the all the θ bypasses.
The destination now modifies its database to incorporate how many channels to restore through which bypasses and sends θ BYPASS-CONF (bypass confirmation) packets; one through each bypass to inform the links on the bypass. The contents of the BYPASS-CONF packet are *link_id*, *bypass_route*, *committed* (the number of channels that will be restored through this bypass) and the *bottleneck* (the bottleneck link on the bypass; same as the *bottleneck* that was written on the corresponding ROUTE-CAPACITY-CHECKER packet). An intermediate node on receiving a BYPASS-CONF packet "remembers" that this link will use me for restoring *committed* more channels. It does so by updating the entry for *link_id* in its *promised channels list*. If needed the intermediate node updates the link's γ.

When the source of the link receives a BYPASS-CONF packet, it updates δ and pushes the *bottleneck* link into the list of bottleneck links for that link. Once the source has received BYPASS-CONF packets for all the θ ROUTE-CAPACITY-CHECKER packets it sent, it again checks if the link is in a critical state. If so, another round of the bypass search procedure is performed. However this time, the bypass route search is done by logically removing the bottleneck links from the topology. On the other hand, if the link is no longer in a critical state, the bottleneck list is flushed.

It could happen that between the source initiated forward directed ROUTE-CAPACITY-CHECKER and the destination initiated reverse directed BYPASS-CONF packets, new paths got set up on some link (u, v) on the bypass. This would reduce the spare capacity of the link (u, v) and could also reduce the spare capacity of the bypass. If such a situation occurs, node u would detect it using the numbers $\beta_{u,v}$ and *committed*, and would send a BYPASS-CONF-NACK packet back to the destination of the link back along the bypass. The contents of a BYPASS-CONF-NACK packet are: the *link_id* of the link for whom we are trying to find bypass routes, the *bypass_route* , and *rejected* (the number of channels that could not be promised = *committed* - β) When a node receives a BYPASS-CONF-NACK packet, it decreases *link_id*'s entry in the *promised channels list* of the proper link by the amount *rejected* and forwards the packet to the next node.

4.3. Kickout and Decrement

Whenever a primary route is being set up through a link L, α_L is incremented by 1 (and obviously β_L gets decremented by 1). In doing so, the difference $(\delta - \alpha)$ might go down below the threshold T and a critical state might be reached. If so, new bypass routes will be searched. Looking at link L as a restoring link, it will have promised a maximum of γ_L channels for restoring other links. After a new connection has been setup through L it could happen

that $\gamma > \beta$ (remember that β is the number of spare channels on a link). Therefore, the link has to inform all links to whom it had promised γ channels for restoration about such a situation. This is done by sending a KICKOUT packet. The contents of a KICKOUT packet are the *originating_link* which initiated the kickout, i.e. L the *affected_link* which is being kicked out (Only those links are kicked out that had been promised *exactly* γ_L channels for restoration by L), *num_kickout* (the number of channels being kicked out $=\gamma - \beta_{new}$), and *route* (a route to the destination end of the affected link). An intermediate node upon receiving a KICKOUT node just forwards it to the next node on the *route* written on the packet. When a destination node of a link L gets a KICKOUT packet it looks at the *restoration table* and selects a set of bypasses containing the *originating_link* among which to distribute the decrease. It then sends DECREMENT packets along these bypasses to inform the links on the bypass about their respective decrease. Obviously the *restoration table* is updated accordingly. The contents of a DECREMENT packet are: *affected_link* (the recipient of the KICKOUT packet), *originating_link* (same as the *originating_link* of the *kickout* packet, *num_decrement* (the decrease of restoration capacity on this bypass; the sum of the *num_decrement* fields for all the DECREMENT packets is equal to *num_kickout* of the KICKOUT packet) and the *bypass*.

An intermediate node on receiving a DECREMENT packet decrements the *promised channels list* entry for the *affected link* by *num_decrement*. It also updates its γ if needed. The *originating_link* link which originally triggered the DECREMENT packet also updates its γ only now. When the source of the affected link receives a DECREMENT packet, besides doing all the things that an intermediate node does, it also decreases δ of the *affected link*, and adds *originating link* to the *bottleneck links list* of the *affected link*. If $\delta_{new} - \alpha < T$, then obviously it has to search for new bypass routes.

5. The Restoration Procedure

The pre-computed restoration routes for each link are stored at the destination ends. Because no reserved capacity is allocated for the backup bypass route, the route cannot be setup beforehand. The actual route setup (the establishment of switching mechanisms at cross connects along the route) can only be performed in real time after a failure occurs. Conceptually the activation procedure should work as follows. Once the destination node detects failure of the service route, it first initiates cross-connect operations on the bypass route and signals the source node to bridge the traffic to this route.

At a more detailed level, the activation procedure requires the following functions, which must be supported by the hardware: failure detection at the destination end of the link, inter-node message exchanges, message processing, cross-connect execution, traffic bridging, and signal selection. We assume that

these functions are supported by the hardware and software architectures. As described in [3] there can be two ways to implement the activation procedure - one a sequential architecture and the other a parallel architecture. ZRESTORE can use either of the two approaches. However, we have used sequential architecture in our simulations since it is more intuitive and easier to understand. The basic idea of the sequential activation architecture has been discussed in [3] and is not being discussed due to space constraints.

Upon detecting that the link has failed, the destination node of the link first determines a list of incoming port numbers for channel_ids say, cid1 to cidx and switches its signal selector to receive signals on these ports. It then determines the corresponding outgoing port number at its preceding neighbor nodes for each incoming port number that it had selected. The node encloses in separate messages to the bypasses, two information items. The first is a table of channel_ids and respective outgoing port numbers for the cross connection to be made. The second is the bypass route. Upon receiving the message, an intermediate node does the necessary switching and forwards the message. This step is repeated, one node at a time, until the source node is reached where the traffic is bridged to the new channels.

6. Simulation

The main goals of our simulations on ZRESTORE were to determine the following performance criteria: The restorability (total_active_restorable_channels / total_active_channels), the control packet overhead and the kind of restoration times provided by ZRESTORE.

We simulated ZRESTORE using `ns2`. The simulation generated a random network topology for a given number of nodes. Links between nodes were generated randomly with a fixed probability for each link. For all our experiments, the probability value was such that each node had an average of 8 links adjacent to it. The values of the paramters were $\lambda_{max} = 16, \theta = 2, T = 2$.

We wanted to determine how well ZRESTORE performed with respect to load in the network. So, connection generation was an important part of our simulation. We first calculate the duration of the whole simulation, any random number between 20s and 40s. We then divide the duration into three phases:

1 Start phase: in this phase, the number of connections in the network rises steadily to a predetermined value.

2 Constant load: an almost constant number but dynamic set of connections is maintained.

3 Finish: the connections just die down gradually.

Connections are explicitly set up and torn down. We just simulate the control plane, and assume connections to be constant bit rate traffic of 10mbps for our

calculations. Each connection has a uniformly distributed length with a mean of 20s. Restoration time is computed by stamping the RESTORE packet with the delays incurred at each node. After every 0.5 seconds we take a snapshot of the network. The metrics we get from these snapshots are:

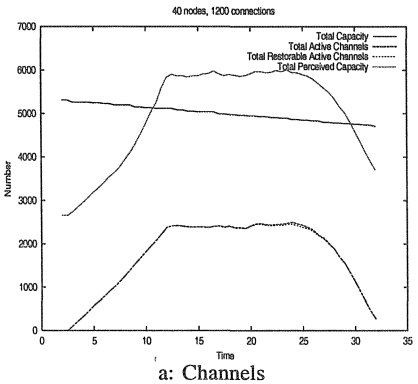
- network capacity: the total available channels in the network
- total active channels: this is the number of channels in use by active paths
- total restorable channels: we define a number for each link, $\delta' = min(\delta, \alpha)$. That is, the number of active channels in the network for which a bypass channel is available. Total restorable channels is the sum of all δ'.
- total perceived capacity: we basically assume that every bypass that a channel perceives is equivalent to an *extra channel* in the network.So for each link, its perceived capacity is $\alpha + \delta$. Total perceived capacity is the sum of this number over all links. This metric gives us an estimate of how much *extra* capacity our scheme is providing to the network.

7. Results

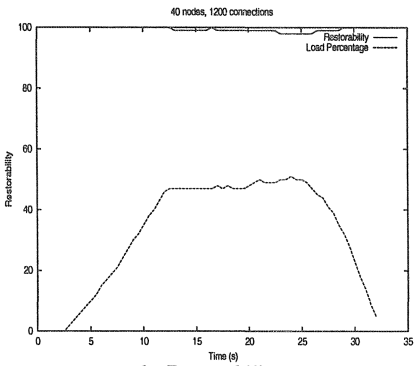
This section talks about the results that we gathered from our simulation runs. We ran simulations for nodes from 10 to 150 in number, with a maximum number of connections ranging from 7 times the number of nodes to 75 times the number of nodes. We found that the pattern of restorability and indeed all the other graphs depends only on the load percentage of the scenario. Therefore the graphs that we present here are the graphs corresponding to 40 nodes subjected to varying load percentages. We put down links for ever, since we assume that these are mechanical faults and thus they are not going to be recovered in the near future. We put down links equal to the number of nodes in the network. When we talk about restoration times, we assume certain constants, some of which have been assumed by authors in [5, 6]. Namely, we have link delay as 400us, node processing delay as 10us, failure determination delay as 500us, cross connecting delay as 500us and traffic bridging delay as 500us. We also calculate the control packet overhead added by our protocol. The control packet size has been taken to be 20 bytes. The other parameters of ZRESTORE assume the values $\lambda_{max} = 16, \theta = 2$ and $T = 2$. We present graphs here for three kind of scenarios, *average load* , *heavy load* and *extreme load.*

7.1. Average Load

Figure 1a shows the situation when there are a maximum of around 1200 connections at a time in the network leading to a load percentage of about 50% in the maximum. We can see that the number of restorable channels remains pretty close to the total number of active channels. This feeling is enforced

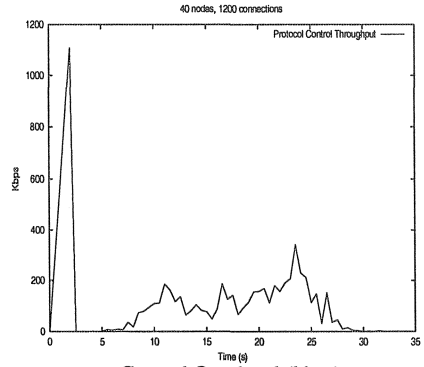


a: Channels

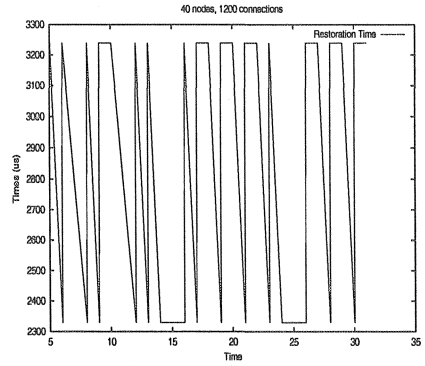


b: Restorability

Figure 1. 40 nodes, 1200 connections



a: Control Overhead (kbps)



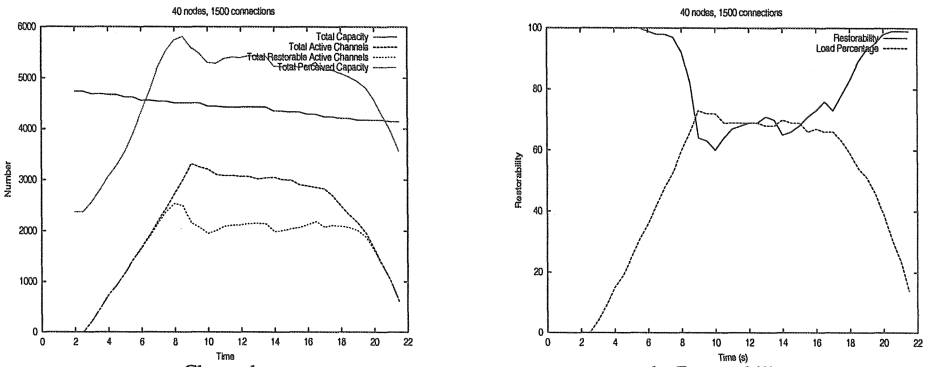
b: Restoration Time

Figure 2. 40 nodes, 1200 connections

by figure 1b in which we find restorability remaining 100% almost throughout dipping to a low of 95%. We then look at the control packet overhead and the kind of restoration times that we get in figure 2. Depending on whether bypasses are 2 hops long or 3, we get a value of 2.33ms or 3.24ms while control overhead has an initial high of 1.1mbps during the setting up of bypasses during network inception, and then hovers in the 200kbps normally.

7.2. Heavy Load

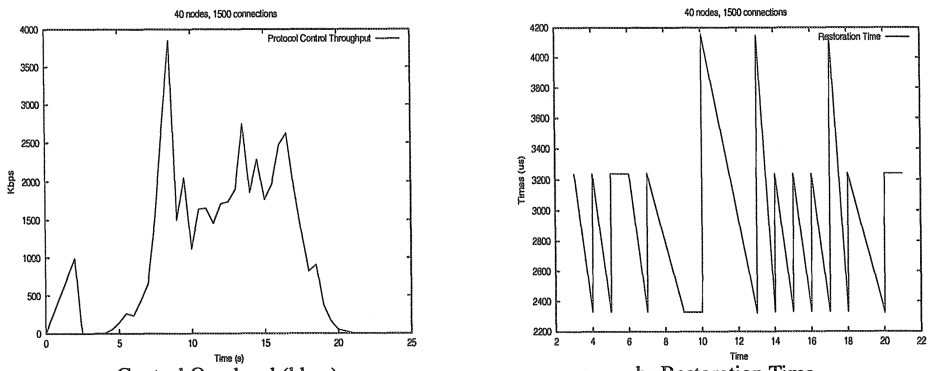
Figure 3a shows the situation when there are a maximum of around 1500 connections in the network leading to a load percentage of about 70% in the maximum. The number of restorable channels remains tolerably close to the total number of active channels.



a: Channels b: Restorability

Figure 3. 40 nodes, 1500 connections

In figure 3b we find restorability dipping to 70% during the load phase, where there is a 70% load on the network. Given that the path selection algorithm is shortest path and no capacity is reserved for restoration, this is a good figure. We then look at the kind of control overhead and restoration times that we get in

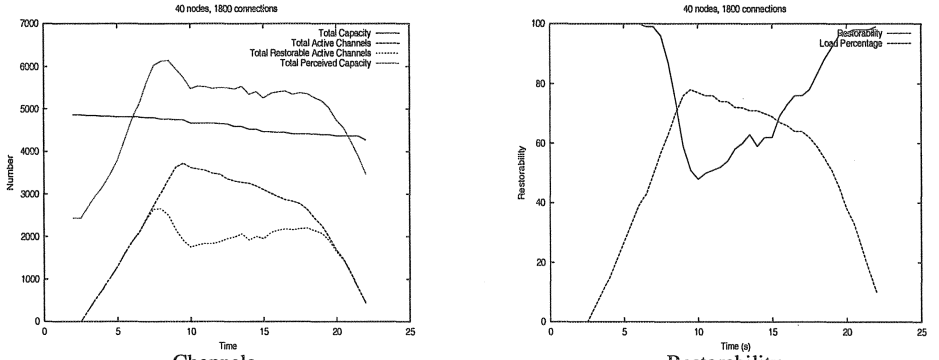


a: Control Overhead (kbps) b: Restoration Time

Figure 4. 40 nodes, 1500 connections

figure 4. Typically bypasses are 2 or 3 hops, so we mostly get a value of 2.33ms or 3.24ms, but sometimes we have bypasses of 4 hops giving a restoration time of around 4.2ms. Control packet overhead initially starts at around 1mbps and then touches around 3.8mbps once during the load stage,staying around the 2mbps at other times.

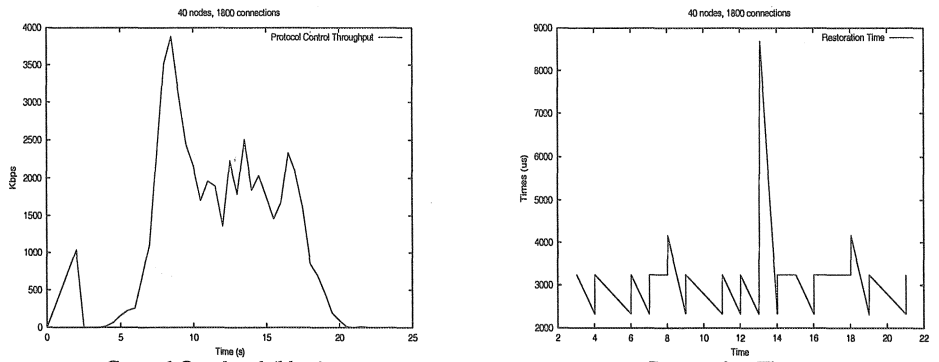
7.3. Extreme Load



Channels Restorability

Figure 5. 40 nodes, 1800 connections

Figure 5a shows the situation when there are a maximum of around 1800 connections in the network leading to a load percentage of about 80% in the maximum. We can see in figure 5b that the number of restorable channels in such extreme load conditions goes down a bit for the load stage to about 50%, and recovers back to full restorability as connection load decreases. We then



Control Overhead (kbps) Restoration Time

Figure 6. 40 nodes, 1800 connections

look at the kind of restoration times that we get in figure 6. Typically bypasses are 2, 3 or 4 hops, so we mostly get a value of 2 to 4ms, in one case we get a value of around 8ms due to a bypass of 8 hops while control overhead in this case again has a maximum of around 4mbps.

8. Extensions to enhance robustness and reliability

We have merely considered ZRESTORE as a restoration scheme, which does not interfere with the path setup mechanism. Thus, ZRESTORE being a non-reservation scheme, always allows calls to come through, even if that means kicking out some promised channels for restoration. However, we can have malicious sources setting up lots of connections and hogging all the channels, thus leaving precious little free, making restoration practically impossible. We would like to avoid such situations. In the optimal case, we would like ZRESTORE to block a call if accepting it would compromise the 100% restorability of the network. This section explores the various algorithms that we can use to enforce some sort of call admission control, thus ensuring that the restorability of the network is not compromised.

The overall idea of the dynamic approach is that a link allows calls to come in until it realizes that the network is now in critical state, i.e., its local neighborhood has links which are not able to restore all of their active traffic. Therefore the link needs to block calls from now on. How the link realises the critical nature of the network and how it decides when to block calls is really protocol dependent. So the realization of a dynamic approach that we propose is ZRESTORE specific.

We propose that a link have three modes, *blind call accepting*, *cautious call accepting* and *call blocking*. At any point, a link will be in one of these modes. When a link is in the *blind call accepting* mode, it will accept all calls just like in normal ZRESTORE. However, when it needs to kickout channels due to calls being set up, it goes into *cautious call accepting* mode, and listens for *Route Capacity Checker* packets coming from the links that have just been kicked out. If the link which has just been kicked out, responds with a *Route Capacity Checker* packet, it means that the link is critical. No response means that the link was over protecting its channels, so this kickout hasn't really affected the active channels on it. Thus we don't need to block calls yet. However, if we receive a response, this means that the link is in critical state. We allow the link one such wave, and do nothing about it. However if we get another *Route Capacity Checker* packet from the same link, we decide that the link is in deep criticality, so from now on we are going to block calls, so we go into *call blocking stage*. All these stages are soft state, that is after staying for some time in *cautious call blocking*, the link goes back to *blind call accepting*, and after some time in *call blocking* it goes back to *cautious call accepting*. A link stays in *call blocking* mode as long as it keeps getting *Route Capacity Checker* packets from the "kicked out" links. If *call blocking* mode continues for a long enough time, we may drop some of the connections on the link. Thus with this scheme, we can ensure a much higher degree of restorability than with an *uncontrolled* network.

9. Conclusions And Future Work

In this paper we have proposed a new scheme for link restoration in optical networks. Our scheme uses dynamic provisioning of non-dedicated bypass routes. We propose the novel idea of intelligently over-protecting channels to avoid searching for a bypass every time a primary channel is set up through a link. For finding bypasses, we perform simultaneous aggregated searches. Simultaneous searches help in finding bypasses quickly and aggregation helps in less overheads duing the search. We intelligently distribute the available channels evenly across the found bypasses.

Using detailed simulation studies we show that our scheme achieves fast restoration times. More importantly, we show that inspite of using a greedy route selection algorithm namely min hop path and not explicitly reserving channels on the bypass route, we maintain a very respectable level of restorability. We also show that the control traffic overhead incurred by our scheme is very minimal especially when conpared to the huge data traffic in optical networks.

For future work, we would like to mention the following problems. As we have stated before, almost all link restoration schemes suffer from their inability to handle node failures. Although path restoration schemes can handle node failures, they are inherently slow. The middle path between these two extremes is segment protection. In this a short segment is protected rather a link or a path. Inventing a segment protection scheme that incorporates the good features of ZRESTORE without compromising on flexibility is an interesting problem.

Many algorithms used in our scheme are actually black boxes and can be replaced by other algorithms. Examples are the algorithms to calculate η, compute the path on which to check for free channels, setting parameters like θ and T and the call admission control. Our scheme might work well for some choices of algorithms and not so well for others. Studying and understanding this is also an interesting project.

Although our scheme performs well under simulation studies, a factor against it is its complexity. Designing a scheme which uses ideas similar to ZRESTORE but is less complex would be very desirable.

References

[1] J. Anderson, J.S. Manchester, A. Rodriguez-Moral, and M. Veeraraghavan. "Protocols and Architectures for Optical IP Networking". *Bell Labs Technical Journal*, Mar. 1999.

[2] C.E Chow, S. McCaughey, and S. Syed. "RREACT: A Distributed Protocol for Rapid Restoration of Active Communication Trunks". In *Second IEEE Network Management and Control Workshop*, 1993.

[3] B. Doshi, S. Dravida, P. Harshavardhana, O. Hauser, and Y. Wang. "Optical Network Design and Restoration". *Bell Labs Technical Journal*, Sep. 1999.

[4] R.R. Iraschko, W.D. Grover, and M.H. MacGregor. "A Distributed Real-Time Path Restoration Protocol with Performance Close to Centralized Multi-Commodity Maxflow". In *1st Intl. Workshop on Design of Reliable Communication Networks*, 1998.

[5] S. Ramamurthy and B. Mukherjee. "Survivable WDM Mesh Networks, Part I–Protection". In *IEEE Infocom '97*, 1997.

[6] S. Ramamurthy and B. Mukherjee. "Survivable WDM Mesh Networks, Part II–Restoration". In *IEEE Infocom '97*, 1997.