Decentralized Local Backup LSP Calculation with Efficient Bandwidth Sharing

Laurent Mélon[*], François Blanchy, Guy Leduc

Research Unit in Networking
Electrical Engineering and Computer Science Department
University of Liège

{melon,blanchy,leduc}@run.montefiore.ulg.ac.be

Abstract – This paper focuses on the protection of virtual circuits (Label Switched Paths, LSPs) in a (G)MPLS [1] (Generalised Multi-Protocol Label Switching) network. The proposed algorithm is designed to protect traffic with strong delay requirements such as EF (Expedited Forwarding) ordered aggregates in a DiffServ domain. For this type of application end-to-end recovery schemes are usually considered to be far too slow. Local, fast-rerouting is the only solution which can compete with restoration times offered by SONET self-healing ring. However local restoration has a cost in terms of extra bandwidth consumed for the backup paths. Our scheme thus includes a sophisticated resource aggregation mechanism based on the concepts of "backup-backup aggregation" and "backup-primary aggregation". The path selection algorithm is also designed to efficiently reduce the resource usage. Moreover, when considering LSPs at different preemption levels, our algorithm is able to correctly calculate the amount of bandwidth that can be preempted despite the sharing of resource. We show that our approach, though local, can compete with the state-of-the-art end-to-end recovery schemes (such as the one presented in [2]) in terms of resource reservation. The major contribution of our scheme, the "backup-primary aggregation", was then also used in the context of end-to-end recovery and improved its performance substantially.

Keywords : Fast recovery, MPLS, resource sharing, backup LSP

1 Introduction

The introduction of optical technology in IP networks and the development of near-terabit capacity routers have made huge amounts of bandwidth available. Simultaneously, the deployment of (G)MPLS and DiffServ in core networks allows ISPs to traffic engineer their network for maximum efficiency and to offer quality of service to their customers. Indeed, MPLS behaves mostly like ATM where VCs (Virtual Circuits) are replaced by LSPs (Label Switched Paths). LSPs can be established to "emulate" the classical hop-by-hop routing of IP but can also be source-routed, allowing to precisely define how flows of traffic must make their way through the network. Signalling protocols associated with MPLS, namely RSVP-TE [3] (ReSerVation Protocol with Traffic engineering Extensions), CR-LDP [4] (Constraint Routing Label Distribution Protocol) both support the provisioning of LSPs with bandwidth allowing to offer end-to-end QoS guarantees.

Optical and electronic hardware running at breath-cutting speeds are inherently unreliable. Hardware failures, software bugs, maintenance/upgrade operations cannot be avoided at some points in time. But, when forwarding packets at the speed of 10Gbps or more, a single second of inactivity means that millions of bytes of precious customer's data are sent directly to the trash-bin and the QoS guarantees you claimed to offer are gone the same way.

Consequently, lots of works have been undertaken to develop algorithms and protocols that will allow the network to quickly recover from any failure it may encounter. Such algorithms where already studied for ATM and before for X.25 for example. But at that time virtual circuits where mostly established by hand based on planning tools and traffic measurement. Nowadays many people agree that, to be really useful, all this process has to be automated. Based on a high level request (defining precisely the bandwidth and QoS requirement of the traffic), the network must be able to compute and establish a path from one ingress router to an egress one and to protect it against failures.

A classical way to reach reliability is to use schemes referred to as 1+1 and 1:1 protection (cf. [5]). For each LSP

[*]Research Fellow of the Belgian National Fund for the Scientific Research (F.N.R.S)

we want to establish, we compute two completely disjoint paths from the ingress to the egress. The best of the two is the primary path, the other is the backup path. In the 1+1 scheme both paths are used simultaneously : all packets are duplicated at the ingress and sent on both paths. The egress node continuously monitors both inputs and selects the "best" one. This way of ensuring protection is of course very costly in terms of bandwidth. [6] presents an approach to limit the cost of the 1+1 protection. In the 1:1 scheme, only the primary path is used to forward packets while the backup path is in "standby" mode. If a failure occurs, a message is sent to the ingress which switch the backup and the primary path. The advantage of the 1:1 solution is that under the single failure assumption significant bandwidth saving can be realized. Indeed if we assume that only a single failure may happen in the network at any given time, not all backup paths can be activated simultaneously. Resources that must be reserved for independent backup paths can thus be shared. For reliability and performance the computation of primary and backup paths will probably be distributed and handled by ingress nodes. Some information must thus be made available to these devices if we want to achieve the best possible aggregation of resources. [7] and [2] present two approaches to do so. The authors of [2] show that with a relatively small amount of data their algorithm is able to protect the network against link failures while limiting the extra bandwidth consumption to 63-68% which is a significant improvement compared to previous methods.

Obviously the 1:1 protection induces far more delay than the 1+1. Failure has to be detected, a message must propagate to the ingress node which must then switch active and standby paths. For this reason other approaches have also been envisaged. In fact restoration strategies can be divided into two classes : end-to-end recovery and local recovery (often called "fast re-routing"). In a local scheme, the re-routing is handled by the node directly preceeding the failure on the primary path or more generally by a node "close" to the failure. The principle is to establish a set of backup LSPs each one protecting the primary path against the failure of one particular node (or link).

It should also be noted that in both local and end-to-end approaches, the computation and establishment of the backup paths can be postponed until the occurrence of a failure. While this way of proceeding makes sense especially with best effort traffic, it introduces large delays and, moreover, restoration cannot be guaranteed to succeed. In the context of this paper we consider only traffics with "strong" protection requirements (the kind of guarantees one may want for voice traffic for example). Without any loss of generality this paper will mainly focus on node failures.

2 Algorithm overview

Our algorithm offers improvements in two main areas compared to any previously presented solution : (1) high quality local bandwidth aggregation and (2) correct handling of preemption levels.

2.1 Bandwidth sharing

2.1.1 Backup-Backup bandwidth sharing

As already explained, resource sharing is possible under the assumption that, at any given time, at most a single failure will occur in the network. If this hypothesis holds, two backup LSPs protecting two distinct nodes will never be activated together. If these LSPs are using some common links, only the maximum bandwidth has to be reserved. Figure 1 presents this scenario. The backup LSP "Src$_1$-Src$_2$-B-C" was established to protect the primary LSP established between Src$_1$ and Dst$_1$ against the failure of node A. Meanwhile, the backup LSP "B-C-Dst$_1$-D-E" was established to protect the primary path between Src$_2$ and Dst$_2$ against the failure of the node labelled "failure". Both backup paths are using the link "B-C" on which we can reserve $max(BW(\text{Backup}_1), BW(\text{Backup}_2))$ where $BW(l)$ represents the bandwidth required by the LSP "l". This type of aggregation allows a high decrease in the bandwidth consumption.

2.1.2 Primary-Backup bandwidth sharing

It is possible to improve further the scheme if we consider that when a backup path is activated because of a failure, some bandwidth isn't used any more on the primary path. Indeed as soon as the failure is detected, the node responsible for the local backup will switch the traffic between service and recovery path. Very rapidly the circumvented links of the primary path will see their bandwidth consumption reduced. This bandwidth can thus be used by other backup LSPs protecting the same node. Figure 1 details the situation. As in the previous case, Backup$_2$ was established to protect LSP$_2$. A second backup was created to protect LSP$_1$ against the failure of the node "failure" : Backup$_3$. Backup$_2$ and LSP$_1$ share link "C-Dst$_1$". If node "failure" fails, packets propagated on LSP$_1$ will be encapsulated into the Backup$_3$ tunnel at node "A". At the same time, packets propagated on LSP$_2$ will be diverted on Backup$_2$. As Backup$_2$ and LSP$_1$ are never used simultaneously, only $max(BW(\text{Backup}_2), BW(\text{LSP}_1))$ must be reserved on link "C-Dst$_1$".

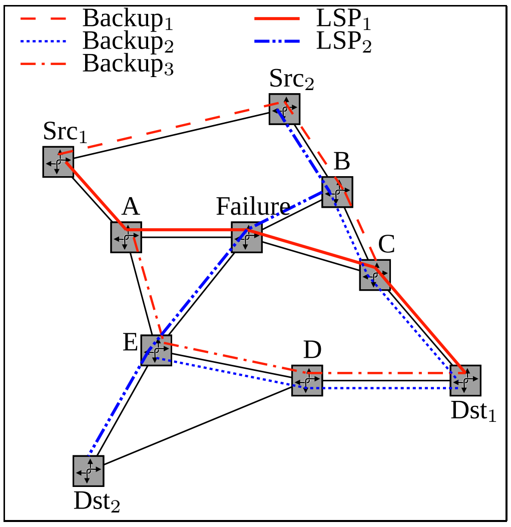

Figure 1: Resources sharing

2.1.3 Path computation

In many traffic engineering schemes, a central server using some sort of mixed-integer programming algorithm computes an optimal mapping between requests and LSP paths. This approach usually require hours of computation and is not very robust mainly because the global optimum discovered by this mean is very sensitive : a few changes in the set of requests leads to a completely different set of paths. In our approach, path computation is completely decentralised and real-time. Establishment requests are processed one after the other by the ingress node from which the LSP must start, each ingress treating the requests it receives independantly. Consequently our scheme can combine easily with TE (Traffic Engineering) algorithms following the same decentralised philosophy, as well as with any more centralised scheme.

In this paper we will always assume that the primary LSP follows the shortest path according to a certain metric (usually a hop count). The reader is referred to [8, 9, 10] for other TE schemes to establish primary paths. When the primary path is known, we compute the set of backup LSPs required to prevent any possible node failure along this path. This is done by starting to protect against last node failure and then going backward until we reach the first node. If a backup path cannot be found under the node-failure hypothesis[1], we assume that only a link failure will occur and compute a new backup path. If it fails again, the request is rejected.

[1]Obviously it is impossible to protect the path against the failure of the egress node. However it is possible to protect the link between the penultimate node and the egress. The backup computed for this purpose only needs to be link-disjoint with the primary path.

To utilise bandwidth efficiently, the path computation algorithm must favour paths where a lot of resource sharing is possible. To compute the backup path we associate with each link a cost corresponding to the increment of bandwidth required if the backup LSP goes through the considered link. Dijkstra's algorithm is then used to compute the shortest path starting at the node preceding the protected node of the primary path towards the egress node. We stop the algorithm when it comes to a node that belongs to the primary path which is located after the protected node.

2.2 Preemption levels

Preemption levels are used to define some LSPs as being "more important" than others. When establishing a LSP, it can preempt the bandwidth reserved by LSPs having a lower preemption level. This makes it possible to request an establishment that will never block. In case of failure, LSPs with a higher preemption level will also be restored first.

Handling preemption levels has no impact on the bandwidth sharing efficiency. However combining resource sharing and preemption levels in the same scheme requires some special care. This problem will be explained more extensively in section 3.2.

3 Detailed description

For the clarity of the rest of this paper we will first define a few terms and functions. While the preemption levels are meaningless for the bandwidth sharing, the most general notation will be introduced. A network is represented by a multi-valued graph $\mathcal{G} = (\mathcal{X}, \mathcal{U})$ where $\mathcal{X}$ is a set of nodes and $\mathcal{U}$ a set of oriented arcs between these nodes. Each link $L_{ij} \in \mathcal{U}$ between nodes $N_i \in \mathcal{X}$ and $N_j \in \mathcal{X}$ is associated with a set of values :

- C_{ij} : the capacity of the link.
- $R_{ij}[p]$: the total bandwidth reserved at preemption level p. $R_{ij} = \sum_{p=0}^{P-1} R_{ij}[p]$.
- $P_{ij}[p]$: the total bandwidth reserved at preemption level p for primary LSPs. $P_{ij} = \sum_{p=0}^{P-1} P_{ij}[p]$.
- $B_{ij}(L_{kn})[p]$: the total bandwidth used by backup LSPs at preemption level p in case of failure of link L_{kn}. $B_{ij}(L_{kn}) = \sum_{p=0}^{P-1} B_{ij}(L_{kn})[p]$.
- $B_{ij}(N_k)[p]$: the total bandwidth used by backup LSPs at preemption level p in case of failure of node N_k. $B_{ij}(N_k) = \sum_{p=0}^{P-1} B_{ij}(N_k)[p]$.
- $F_{ij}(L_{kn})[p]$: the total bandwidth freed by primary LSPs at preemption level p in case of failure of link L_{kn}. $F_{ij}(L_{kn}) = \sum_{p=0}^{P-1} F_{ij}(L_{kn})[p]$.

- $F_{ij}(N_k)[p]$: the total bandwidth freed by primary LSPs at preemption level p in case of failure of node N_k. $F_{ij}(N_k) = \sum_{p=0}^{P-1} F_{ij}(N_k)[p]$.

P is the number of preemption levels. In a practical implementation the source node of each arc is responsible for maintaining this set of values up-to-date. From these definitions, we have :

$$B_{ij}(N_k)[p] = \sum_{\forall m: L_{mk} \in \mathcal{U}} B_{ij}(L_{mk})[p] \tag{1}$$

$$F_{ij}(N_k)[p] = \sum_{\forall m: L_{mk} \in \mathcal{U}} F_{ij}(L_{mk})[p] \tag{2}$$

$$R_{ij} = P_{ij} + \max\left(0, \max_{L \in \mathcal{U}} \left(B_{ij}(L) - F_{ij}(L)\right), \max_{N \in \mathcal{X}} \left(B_{ij}(N) - F_{ij}(N)\right)\right) \tag{3}$$

Note that in equation 3 we have to consider the maximum over all possible link failure scenarios even if we are protecting against node failure because it is not mathematically guaranteed that the worst case bandwidth consumption will be obtained when faced with a node failure. Indeed consider the failure of link B-Failure on figure 1. In this scenario, both Backup$_2$ and LSP$_1$ will be used simultaneously while it is not the case if node “Failure” goes down. Of course, in most pratical situation the worst case will be a node failure.

A LSP request is composed of :

- the source or ingress node : src ;
- the destination or egress node : dst ;
- the required bandwidth[2] : bw

3.1 Path computation

3.1.1 Primary path computation

Dijkstra’s algorithm [12] is used to find the shortest path from src to dst. The resulting path can be described by an ordered set $\mathcal{P} = \{N_{x_0}, N_{x_1}, \ldots, N_{x_n}\}$ with $N_{x_0} = src$ and $N_{x_n} = dst$.

[2] In this paper we assume that a single value defines the bandwidth required by each LSP. In a DiffServ context this corresponds to using L-LSPs or E-LSPs with a single OA (Ordered Aggregate). Extensions of the presented algorithms to handle E-LSPs with multiple OAs is straightforward and will be presented in our future works. Interested readers are invited to read [11] for further information on how to combine DiffServ and (G)MPLS and for a definition of L-LSPs and E-LSPs.

3.1.2 Backup paths computation

Given the primary path $\mathcal{P}$ we do :

1. $k \leftarrow n$
2. if $k > 0$ then $k \leftarrow k - 1$ else terminate
3. The node we try to protect is $F = N_{x_{k+1}}$
4. Each link is assigned a cost K_{ij} given by
   - if we protect against node failure

$$K_{ij} = \begin{cases} Inc_{ij}(F, bw) & \text{if } i \neq F \wedge j \neq F \\ & \wedge Inc_{ij}(F, bw) \neq 0 \\ \varepsilon & \text{if } i \neq F \wedge j \neq F \\ & \wedge Inc_{ij}(F, bw) = 0 \\ \infty & \text{if } i = F \vee j = F \end{cases}$$

   - if we protect against link failure

$$K_{ij} = \begin{cases} Inc_{ij}(F, bw) & \text{if } (i \neq N_{x_k} \vee j \neq F) \\ & \wedge Inc_{ij}(F, bw) \neq 0 \\ \varepsilon & \text{if } (i \neq N_{x_k} \vee j \neq F) \\ & \wedge Inc_{ij}(F, bw) = 0 \\ \infty & \text{if } i = N_{x_k} \wedge j = F \end{cases}$$

   Where $Inc_{ij}(F, bw)$ represents the increase of R_{ij} when a backup LSP requiring bw units of bandwidth and protecting node F uses link L_{ij}. We have $Inc_{ij}(F, bw) = R_{ij}^{'} - R_{ij}$ where $R_{ij}^{'}$ denotes the new reserved bandwidth obtained after the new LSP establishment. If $R_{ij}^{'} > C_{ij}$ than $Inc_{ij}(F, bw) = \infty$ (capacity constraint). It can be calculated using equations 1, 2 and 3. Dijkstra’s algorithm is run from root N_{x_k} until the next marked node by the procedure is N^* with $N^* \in \mathcal{P}$. If no valid node-disjoint path is found, then return to step 4 and select link failure protection. If it fails once again, reject the request. ε is a small number which is used instead of zero to favour the selection of the shortest path if all Inc_{ij} are null.
5. go to step 2.

This algorithm can even be further distributed to spread the load of computation across as many routers as possible. Indeed each router along the primary path could be responsible for computing and establishing the backup LSP protecting the next node. This approach would require some sort of synchronisation as, to benefit from maximum sharing, backup paths cannot be calculated independently. It’s worth also noting that this procedure is not optimal for two reasons. First of all, it is not a network-wide optimum. It is quite obvious because requests are treated one after the other. This means that choices made for any particular LSP will never be re-evaluated in the future. But this procedure is also not optimal at the LSP level, i.e. does not lead to find the set of LSPs minimising the increase of bandwidth reservation. Indeed backup LSPs are calculated in order starting at the last node and going backwards. Once again the imposed order prevents the algorithm to find an optimal solution.

Despite being sub-optimal, during our simulations, this algorithm proved to be a good heuristic. We also tested an enhanced version of the algorithm which consists of iterating on backup paths calculation until no more gain is observed. In very rare cases it leads to a global improvement on the total bandwidth consumed at the network level. In all other cases it leads to no improvement at all or even to an increase in the bandwidth consumption.

Moreover the type of solution we propose is designed to be used also in a very dynamic environment where relatively small LSPs (compared to the links capacity) are added and removed permanently, following users needs. In this context, a form of statistical multiplexing makes the ordering of establishment less relevant.

3.1.3 Link state management

Each node must maintain and update the link state information for all links it is responsible for. When a new LSP is established through the link L_{ij}, $P_{ij}[p]$, $B_{ij}(L)[p]$ and $F_{ij}(L)[p]$ have to be updated. $R_{ij}[p]$ can then be recomputed by means of a procedure described in section 3.2. Let l be a new LSP of preemption level p_l with $BW(l) = bw$. The path of LSP l is the ordered set $\mathcal{P}_l = \{N_{x_0}, N_{x_1}, \ldots, N_{x_n}\}$. If l is a backup LSP, let its primary path be $\mathcal{P}_p = \{N_{y_0}, N_{y_1}, \ldots, N_{y_m}\}$. Let s be the index for which $N_{y_s} = N_{x_0}$, the protected node being $N_{y_{s+1}}$. Let e be the index for which $N_{y_e} = N_{x_n}$ i.e. the node where primary and backup paths merge. Figure 2 details the situation.

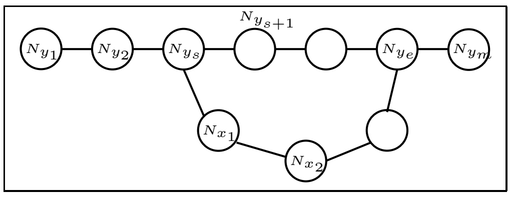

Figure 2: Primary and backup paths

- If l is a primary LSP, links $L_{ij} \in \mathcal{P}_l$[3] must be updated according to :
  1. $P_{ij}[p_l] \leftarrow P_{ij}[p_l] + bw$
  2. $F_{ij}(L)[p_l] \leftarrow F_{ij}(L)[p_l] + bw \quad \forall L : L \in \mathcal{P}_l$

  The purpose of point 2 is to reflect the fact that at this stage of the establishment no backup exists. If a failure occurs at this point, reservations should thus be freed along the primary path. This assumes that when an unprotected LSP is interrupted by a failure the bandwidth reserved is freed after *and* before the failure. This can be achieved quite easily. Indeed each backup LSP is associated with one particular node failure. When a packet flows along a backup LSP, a router can thus know that a failure occurred and where it is located. As the router also knows the path of each primary LSP established through him, it can decide to discard packets on failed LSPs *before* they reach the failure.

- If l is a backup LSP, all links $L_{ij} \in \mathcal{P}_l \cup \mathcal{P}_p$ must be updated according to :
  1. $B_{ij}(L_{y_s y_{s+1}})[p_l] \leftarrow B_{ij}(L_{y_s y_{s+1}})[p_l] + bw$ if $L_{ij} \in \mathcal{P}_l$
  2. $F_{ij}(L_{y_s y_{s+1}})[p_l] \leftarrow F_{ij}(L_{y_s y_{s+1}})[p_l] - bw$ if $(L_{ij} \in \mathcal{P}_p) \wedge (j \le s \vee e \le i)$

  The purpose of point 2 is to reserve bandwidth on the primary path before the protected node and after the merging point. Indeed when the primary LSP was established it was unprotected and this bandwidth was supposed to be freed in case of failure. Now that this recovery path is in place a failure of node "$N_{y_{s+1}}$" is not fatal for the primary LSP.

  Because of point 2, the establishment of a backup LSP must also be signalled on the primary path to allow the reservation of the required resources. In fact the only information that must be made available is the starting and the merging point.

[3]The notation $L_{ij} \in \mathcal{P}_l$ denotes $\exists t : N_{x_t} = N_i \wedge N_{x_{t+1}} = N_j$.

3.2 Preemption levels aggregation

When combining both preemption levels and resources sharing we must be careful that $R_{ij}[p]$ must correctly reflect the amount of bandwidth which can effectively be preempted (if required) at each preemption level. Indeed the preemption level is a property of LSPs not of bandwidth. But we assign an amount of bandwidth to each preemption level to reflect the fact that by removing all LSPs at a given level a certain amount of bandwidth will be freed. When dealing only with primary path everything is very easy : we have to reserve at level p the sum of the bandwidth required by all LSPs at level p. The introduction of backup LSPs and bandwidth sharing makes things a bit more complex. Indeed when using protection, removing a LSP does not necessarily free any resource : if we recall equation 3, we see that a decrease of $B_{ij}(N)$ only has an impact on R_{ij} if node N is the one that maximises the difference. The consequence is that the preemption of a given quantity of bandwidth will sometimes require that we tear down a set of LSPs whose total bandwidth is bigger than the required bandwidth. To do so the LSPs we are establishing must have a preemption level higher than all the LSPs in this set.

An example is given in tables 1 and 2. The bandwidth that must be reserved for backup LSP$_1$ and LSP$_2$ can be limited to $max(BW(\text{LSP}_1), BW(\text{LSP}_2)) = 10$Mbps because they protect two distinct nodes. A new LSP with preemption level 1[4] would only be able to preempt bandwidth from LSP$_1$. But despite the fact that LSP$_1$ requested 10 Mbps of bandwidth, removing it will only free 5 Mbps because of sharing.

LSP	Failure	Bandwidth	Preemption Level
1	N_x	10 Mbps	2
2	N_y	5 Mbps	1

Table 1: Sharing with preemption levels : LSPs

Preemption level	Bandwidth
$R_{ij}[1]$	5 Mbps
$R_{ij}[2]$	5 Mbps
R_{ij}	10 Mbps

Table 2: Sharing with preemption levels : $R_{ij}[p]$

As explained earlier preemption levels are used to give priority to certain LSP requests. If a link is completely filled then it is still possible to establish a new LSP through this link by preempting resources belonging to less important LSPs. But the bandwidth reserved on a link is the result of three terms (cf. equation 3) which are composed in different proportions for each preemption level.

The algorithm computing $R_{ij}[p]$ is composed of two phases. The first one consists of computing an intermediate result $G_{ij}(L)[p]$ and $G_{ij}(N)[p]$. The second phase computes $R_{ij}[p]$ using this result.

3.2.1 Phase 1

The value $G_{ij}(L)[p]$ (resp. $G_{ij}(F)[p]$) represents, up to preemption level p, the increment of bandwidth that must be reserved to be able to forward traffic in case of failure of link L (resp. node F).

$$G_{ij}(L)[p] \leftarrow \max\left(0, \sum_{k=0}^{p} B_{ij}(L)[k] - \sum_{k=0}^{p} F_{ij}(L)[k]\right) \quad \forall p, L : 0 \le p < P, L \in \mathcal{U}$$

$$G_{ij}(F)[p] \leftarrow \max\left(0, \sum_{k=0}^{p} B_{ij}(F)[k] - \sum_{k=0}^{p} F_{ij}(F)[k]\right) \quad \forall p, F : 0 \le p < P, F \in \mathcal{X}$$

The algorithm is based on the following idea : we do not need to reserve extra bandwidth at level p if, up to that level, a sufficient amount of bandwidth will be freed by the failure we consider. However we should note that $G_{ij}(X)[p]$ (X being either a node or a link) can never be negative even if $\sum F_{ij}(X)[p] > \sum B_{ij}(X)[p]$ because it would mean we have to "unreserve" bandwidth that is used by active primary paths (recall that $F_{ij}(X)[p]$ is already reserved).

Figure 3 shows the situation. Up to level 1, a failure will free more bandwidth that needed by the backup LSPs. For this reason $G_{ij}(X)[0] = G_{ij}(X)[1] = 0$. For $p \ge 2$, $\sum_{k=0}^{p} B_{ij}(X)[k] - \sum_{k=0}^{p} F_{ij}(X)[k] > 0$. The values of $G_{ij}(X)[2]$, $G_{ij}(X)[3]$ and $G_{ij}(X)[4]$ are represented graphically on the figure.

$G_{ij}(X)[p]$ is the balance up to level p between the required backup bandwidth and the freed primary bandwidth. If $G_{ij}(X)[p_0] > 0$ for a particular p_0, this means that we must add a new reservation of bandwidth at level p_0 to correct the difference. If we assume that the correction has already been done for all $p < p_0$, the new reservation at level p_0 must consist of $G_{ij}(X)[p_0] - G_{ij}(X)[p_0 - 1]$ units of bandwidth.

This is only for a particular failure. As any node in the network can fail, we have to define a new vector $M_{ij}[p]$ accounting for the maximum difference between the total bandwidth required and freed considering all possible failures. This is the purpose of phase 2.

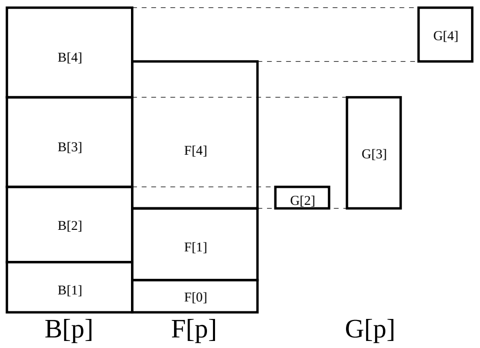

Figure 3: Preemption level selection

[4]Preemption levels are numbered in decreasing order of priority. Level 1 is thus more important than level 2.

3.2.2 Phase 2

We introduce the vector $M_{ij}[p]$ given by :

$$M_{ij}[p] \leftarrow \max\left(\max_{L\in\mathcal{U}}\left(G_{ij}(L)[p]\right), \max_{F\in\mathcal{X}}\left(G_{ij}(F)[p]\right)\right) \qquad \forall p : 0 \leq p < P$$

This vector plays the same role as $G_{ij}(X)[p]$ but at the network-wide level. Now that we have such a failure independent value we can compute :

$$\begin{cases} R_{ij}[0] \leftarrow P_{ij}[0] + M_{ij}[0] \\ R_{ij}[p] \leftarrow P_{ij}[p] + M_{ij}[p] - M_{ij}[p-1] \\ \qquad\qquad \forall p : 0 < p < P \end{cases}$$

Which is the same formulae we introduced by means of our example. It should be pointed out that the difference $M_{ij}[p] - M_{ij}[p-1]$ can be negative which looks a bit surprising at first. Indeed it means we have to reserve less bandwidth at level p than the sum of the bandwidth requirements of all primary LSPs. In fact this just means that a certain amount of bandwidth initially reserved at level p has been upgraded to level $p_0 < p$ to be aggregated with backup LSPs.

3.3 Signalling extensions

As all computations are done by ingress nodes, information regarding the state of each link has to somehow propagate to these nodes. This is done through a signalling protocol which in our case can be of two kinds : a routing protocol or a label distribution protocol.

An easy approach is to flood R_{ij}, $B_{ij}(L) - F_{ij}(L)$ along with a routing protocol PDU. We could extend OSPF to flood this vector regularly to allow ingress nodes to stay permanently up-to-date with the network state. If the TE algorithm used for primary path is required to know the bandwidth usage *per preemption level*, $R_{ij}[p]$ ($\forall p : 0 \leq p < P$) must replace R_{ij}. The delay between two floodings has to be carefully chosen to minimise the number of LSP establishment failures due to obsolete information. This represent a big amount of data. A first way to reduce this would be to flood $B_{ij}(N) - F_{ij}(N)$. If we recall equation 3, we see that this is sufficient if the node failure is always worse that a single link failure (which is a very realistic assumption).

In a very big network, even this “reduced” flooding might became quite costly. The approach followed in [2] is to extend RSVP-TE or CR-LDP to collect information while establishing the primary path. However, to be able to do so, a slightly bigger amount of data must be kept for each link. Each node has to know the impact on the bandwidth consumed on all links L_{ij} in the whole network if each of its links fails. A node N_x must thus collect and maintain the following information :

$B_{ij}(L_{kn})[p]$ and $F_{ij}(L_{kn})[p]$
$\forall i, j, k, n, p : L_{ij} \in \mathcal{U}, k = x \vee n = x, 0 \leq p < P$

The information required for backup path calculation can be easily collected at each primary LSP establishment. Indeed if we use RSVP-TE for LSP signalling, a PATH message will be forwarded along the primary path towards the egress node. When the RESV message is forwarded back to the ingress node, each node N_x belonging to the primary path can append $B_{ij}(N_x) - F_{ij}(N_x)$ to this message. The ingress node will thus receive a fresh network state before calculating the backup paths. R_{ij} should continue to be flooded by a routing protocol. We should note however that this scheme introduces delays since the ingress node must wait for the primary path to be completely signalled to start computing the backup LSPs. Moreover it cannot be used if the computations of the primary and of the backup paths are done together to further minimise resource consumption.

4 Simulations

We used for our tests a topology which was composed of 50 nodes and 102 full-duplex links. Among the 50 nodes, 30 were chosen to act as border routers. We affected to each ingress-egress pair a probability which was used to engineer the network by means of a multi-commodity maxflow algorithm. The purpose of this algorithm is to ensure that if a set of requests follows the given distribution probability a load of 100 % is achievable on all link. This approach was used to mimic the way a real network is engineered. Network engineering in the context of fault protection is still a very active domain of research (cf. [13, 14]).

The most important value when designing a restoration scheme is the “cost” of such a protection in terms of additional bandwidth reserved for backup LSPs. We will call it “network over-subscription” and represent it by γ. It is given by :

$$\gamma = \frac{\sum_{L_{ij}\in\mathcal{U}} R_{ij} - \sum_{L_{ij}\in\mathcal{U}} P_{ij}}{\sum_{L_{ij}\in\mathcal{U}} P_{ij}}$$

γ measures the network-wide bandwidth reservation increase caused by the backup LSPs compared to the unprotected case.

Four algorithms for node-failure protection are presented in the following results. The first one (labelled “LOCAL”) is as the name suggest a local recovery scheme using only the “backup-backup” bandwidth sharing scheme. The second one, “LOCAL with FBW”, is an enhanced version taking into account the concept of “primary-backup aggregation” (using the vector $F_{ij}(F)$). The same difference exists between the two algorithms labelled respectively “END-TO-END” and “END-TO-END with FBW”. It should be noted that the “END-TO-END” algorithm is similar in its behaviour to the algorithm presented in [2] which we consider to be the state-of-the-art in resource aggregation.

4.1 Results

Figure 4 presents the evolution of γ when we progressively add LSPs to the network. The vertical position of each algorithm is not a real surprise. However it should be noted how the introduction of vector F improves the performance of the local recovery. Local restoration with FBW is as good as end-to-end recovery without FBW. On this topology, the “distance” between the best local approach and the best end-to-end scheme is less than 10%, a price we consider quite cheap to benefit from very short restoration delays.

The decrease of γ that occurs after the establishment of 2000 primary LSPs should be pointed out. This behaviour is due to the method to choose the path of the primary LSP. Indeed, as the primary always follows the shortest path, many primary LSPs will tend to overlap while enough bandwidth remains available on the shortest path. This tends to create regions where links are used to protect only a small number of nodes. This situation does not create a lot of opportunities to aggregate bandwidth. If we take a look at figure 5, we will see that after 2000 establishments, the mean reserved bandwidth is close to 45%. This mean load suggests that some links are completely filled. The following requests thus have to make a detour to avoid the saturated links. Because of this, backup LSPs are now established in other parts of the network where an important sharing can be realized.

This is confirmed by figure 6 which shows the mean number of non-null elements in the vector $F_{ij}(F)$, i.e. the evolution of $\frac{1}{|\mathcal{X}|}\sum_{F\in\mathcal{X}} sign(F_{ij}(F))$[5]. Despite not being shown in this paper the same kind of behaviour is observed for the density of vector $B_{ij}(F)$. The slope of the curve increases shortly after 2000 LSP establishments indicating that backup LSPs are now using links where no bandwidth has been reserved for protecting the same node. This suggests that choosing our primary paths in a smarter way could have a big impact on the amount of sharing.

While the absolute value we obtain for the over-subscription cannot be considered a reference value without extensive testing on many different topologies and traffic matrices, it proves the interest of including restoration mechanism in the MPLS layer. Indeed lower layer recovery schemes such as SONET self healing rings impose an over-subscription of 100% and is limited to link-failure protection.

[5] $sign(x)$ is equal to 1 if x is positive, to -1 if x is negative and to 0 if x is equal to 0.

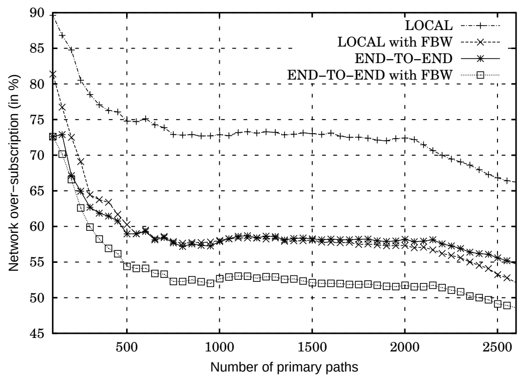

Figure 4: Evolution of network over-subscription

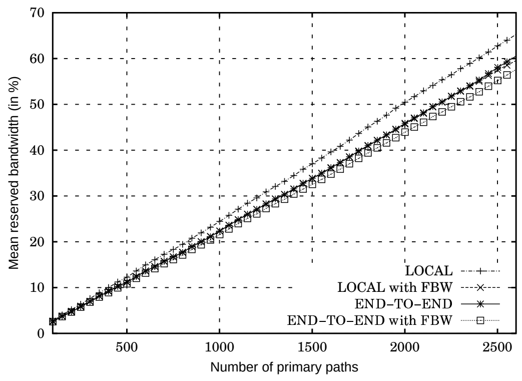

Figure 5: Evolution of mean load

5 Conclusions and future works

The contribution of this paper is twofold. First of all we improved the best known bandwidth sharing scheme without sacrificing simplicity. This new aggregation technique is able to provide a substantial decrease of the network over-subscription of fast-rerouting. We also proposed an efficient way of computing backup paths in a decentralised or even completely distributed manner. The signalling extensions required to implement our approach remains simple and not too costly in terms of resource. The second interest of this paper is to explain the modification required to handle correctly the notion of “preemption levels”.

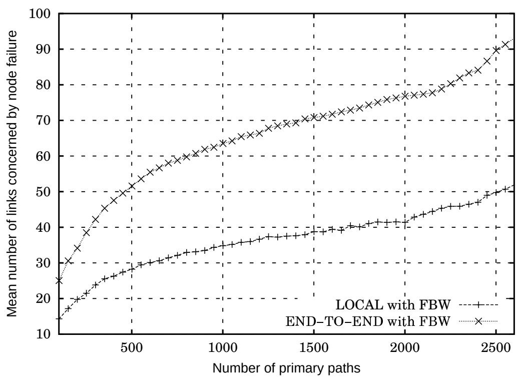

Figure 6: Evolution of the mean number of links having some bandwidth reserved for a particular failure

Results presented show that fast-rerouting is a viable approach to protect traffic that can only accommodate very short interruptions. They also suggest that routing the primary path in a smarter way could help reduce the resource usage further. This topic will probably be an active domain of research for our future works.

Acknowledgement

The authors would like to express their gratitude to Stefaan De Cnodder from Alcatel for his very interesting comments.

This work has been carried out in the IST-1999-20675 ATRIUM project funded by the European commission.

References

[1] Eric C. Rosen, Arun Viswanathan, and Ross Callon. Multiprotocol label switching architecture, rfc3031. www.ietf.org, January 2001.

[2] Guangzhi Li, Dongmei Wang, Charles Kalmanek, and Robert Doverspike. Efficient distributed path selection for shared restoration connections. In *Proceeding of IEEE INFOCOM 2002*, volume 1, 2002.

[3] D. Awduche and al. Applicability statement for extensions to RSVP for LSP-tunnels, rfc3210, December 2001.

[4] B. Jamoussi and al. Constraint-based LSP setup using LDP, rfc3212, January 2002.

[5] Vishal Sharma, Ben-Mack Crane, Srinivas Makam, Ken Owens, Changcheng Huang, Fiffi Hellstrand, Jon Weil, Loa Anderson, Bilel Jamoussi, Brad Cain, Seyhan Civanlar, and Angela Chiu. Framework for MPLS-based recovery, draft-ietf-mpls-recovery-frmwrk-06.txt. Internet-Draft, July 2002.

[6] Koushik Kar, Murali Kodialam, and T.V. Lakshman. Routing restorable bandwidth guaranteed connections using maximum 2-route flows. In *Proceeding of IEEE INFOCOM 2002*, volume 1, 2002.

[7] Murali Kodialam and T.V. Lakshman. Dynamic routing of locally restorable bandwidth guaranteed tunnels using aggregated link usage information. In *Proceeding of IEEE INFOCOM 2001*, volume 1, 2001.

[8] F. Blanchy, L. Mélon, and G. Leduc. An efficient decentralized online traffic engineering algorithm for MPLS networks. Submitted paper available on http://www.run.montefiore.ulg.ac.be/~blanchy, July 2002.

[9] Murali S. Kodialam and T. V. Lakshman. Minimum interference routing with applications to MPLS traffic engineering. In *INFOCOM (2)*, pages 884–893, 2000.

[10] Richard Rabbat, Kenneth P. Laberteaux, Nirav Modi, and John Kenney. Traffic engineering algorithms using MPLS for service differentiation. In *ICC (2)*, pages 791–795, 2000.

[11] F. Le Faucheur and al. Multi-protocol label switching (MPLS) support of differentiated services, rfc3270. www.ietf.org, May 2002.

[12] Dijkstra E. W. A note on two problems in connection with graphs. Numerische Mathematik, 1959.

[13] Yu Liu, David Tipper, and Peerapon Siripongwutikorn. Approximating optimal spare capacity allocation by successive survivable routing. In *Proceeding of IEEE INFOCOM 2001*, pages 699–708, Anchorage, AL, April 24–28 2001.

[14] Yu Liu and David Tipper. Spare capacity allocation for non-linear cost and failure-dependent path restoration. In *Proceeding of the Third International Workshop on Design of Reliable Communication Networks, DRCN 2001*, Budapest, Hungary, October 7–10 2001.