An Efficient Recovery Scheme for Node and Link Failures in Real-Time IP Communications

B Purushotham, Ch D V Subba Rao, *Member, IAENG*, and N Padmaja

***Abstract* — Distributed real-time applications such as medical imaging, traffic control, and video conferencing demand hard guarantees on the message delivery latency and the recovery delay from component failures. In order to meet these requirements there exist number of schemes by reserving additional resources a priori along a backup channel which is disjoint with the primary path. A new method has been proposed for computing segmented backup for recovering from node and link failures in real-time IP communications. This paper compares the performance of proposed scheme with the schemes available in the literature for mesh and other topologies. It has been demonstrated that the proposed scheme gives better results on parameters like number of backup segments, percentage reuse of primary path and average hop count for failure notification.**

***Index Terms* — IP Networks, Fault Tolerance, Segmented Backup, Resilient Routing Layer, QoS with restoration.**

I. INTRODUCTION

IP networks were initially built to handle traffic in a best-effort manner. Best-effort delivery means that IP does its best to transmit packets from source to destination but does not provide any guarantees. But, newly emerging applications demand performance requirements like less delay, high throughput, etc. Quality-of-Service (QoS) in internet is provision of these performance guarantees to the applications. However, failure of a node or link along the transmission path can disrupt the data flow. QoS cannot be achieved in such circumstances. Ensuring seamless flow of data even when failures occur in the network is called as reliability. To achieve reliability, various fault tolerant schemes are used, which are primarily based on computing alternate paths. These backup paths are activated whenever there is a fault. The proposed scheme handles the failure recovery in real time applications by meeting the strict delay constraints.

The rest of this paper is organized as follows: Section 2 presents the literature review of various fault tolerant schemes for real-time IP applications. Section 3 describes the proposed scheme. Performance evaluation/ comparison of the proposed scheme and other existing schemes have been done in section 4. Section 5 presents the conclusions and future enhancements.

Manuscript received April, 2009.

Mr B Purushotham is with Dept. of Computer Science and Engg, C R Engineering College, Renigunta Road, Tirupati – 517 501, India. (phone: +919912211341, e-mail: bpurush_mtech@yahoo.co.in)

Dr Ch D V Subba Rao is with Dept. of Computer Science and Engg., S V University College of Engineering, Tirupati – 517 502, India. (e-mail: subbarao_chdv@hotmail.com)

Mrs N Padmaja is with Dept. of Electronics and Communication Engg., C R Engineering College, Renigunta Road, Tirupati – 517 501, India. (e-mail: padnaja2000@yahoo.co.in)

II. RELATED WORK

In this section, the work done towards finding pre-computed backup paths for achieving routing resilience is reviewed. This section discusses the two significant schemes related to fault tolerance for the internet traffic. They are segmented backup [1] which is particular to real-time applications and resilient routing layers [2] which is for general IP traffic. In this section, we also discuss another scheme studied in this context i.e. QoS paths with restoration [5].

A. Overview of Segmented Backup

In the reactive schemes for fault tolerance there are two different approaches for the computation of backup paths. First approach is end-to-end backup [6] in which backup paths are disjoint with respect to the primary path where primary path is the best path with respect to the criteria of the application. The data is transmitted on this path when no fault has occurred. In the second approach, the primary path is divided into many overlapping sub-paths. Alternate path computation is done by computing backups for these sub-paths. Thus, recovery from faults in this scheme is confined to switching to the backup for the part of the primary path where the failure has occurred. Whenever there is a fault, the corresponding alternate path is activated to re-route the data. This approach is called local detouring [7]. Segmented backup [1] is a kind of a local detouring scheme in which multiple backup paths exist, each protecting a portion of the primary path.

A.1 Illustration of Segmented Backup

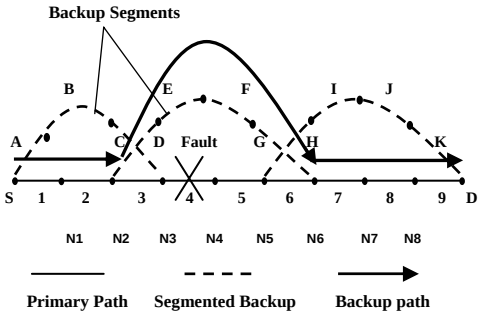

Fig 1. Illustration of a primary path with a segmented backup.

In segmented backup, backups are computed for portions of the primary path. These portions are called as segments. A backup path is found for each segment, which is called as backup segment. All such computed backup segments are collectively referred to as segmented backup. Fig.1 shows an illustration of a primary path with intermediate nodes *N1 to N8* and links 1-9 and its segmented backup. The backup

links are shown as *A–K*. The primary path has three segments, each with its own backup segment. The segments span links 1-3, 3-6 and 6-9, while their corresponding backup segments span links *A-C*, *D-G*, and *H-K*, respectively. These three backup segments together constitute the segmented backup for this primary path. Note that successive segments of a primary path overlap on at-least one link. This is to ensure that there is no single point of failure and recovery is possible for any node failures.

Consider for example that link 4 has failed. Restoration of connection is done by bypassing the segment 3–6 by using its corresponding backup segment D–G. Rest of the primary path is used as it is. Here the recovery is done locally and also there is as much reuse of the allocated resources along the primary path as possible. Local recovery and reuse of the primary path are the fundamental advantages of segmented backup over an end-to-end backup.

A.2 Min_SegBak Algorithm:

Min_SegBak algorithm [1] is used to compute segmented backup. The steps involved in the algorithm are illustrated through an example in the Fig 2. Given a graph *G=(V,E),* construct a directed auxiliary graph *G'(V,E')* using the following rules.

• Auxiliary graph construction.

For every link l=(u,v), where l does not belong to primary path P, add two directed links (v,u) and (u,v) to G' . The cost and the delay of these directed links remains the same as those of the original link as shown in Fig. 2(a). Replace every link l=(u,v) where l belongs to the primary path P with l=(v,u) in G' and assign it zero cost. The links belonging to the primary path are now reversed as shown in Fig. 2(b).

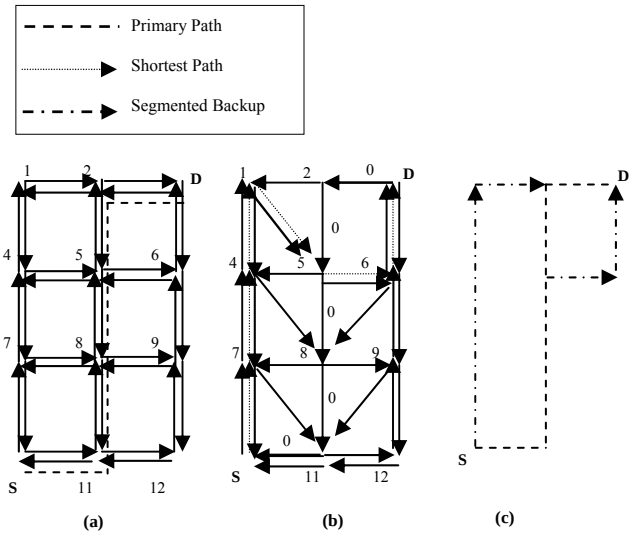

Fig. 2. Illustration of Min_SegBak algorithm. (a) Graph G with Primary path S to D. (b) Modified Graph G' with shortest path S to D. (c) Primary path with backup segments.

For every link l=(u,v) where v belongs to primary path P, replace it with link (u,w) where 'w' is immediate predecessor of 'v' in P as shown in Fig. 2(b).

- Least delay path in the auxiliary graph: Find shortest path P' from source (S) to destination (D) in the auxiliary graph G' shown in Fig. 2(b).
- Construction of segmented backup: Using the primary path and the shortest path in the auxiliary graph the segmented backup is computed vide Fig. 2(c).

B. Resilient Routing Layers (RRL)

RRL [2] is a network resilience scheme meant for handling of general IP traffic. We first describe the terminology associated with this scheme and followed by recovery mechanism.

B.1 Terminology: RRL is based on computing redundant subsets of the network topology which are called layers. Let *G=(V,E)* be the graph representing the network topology, where *V* is the set of vertices representing nodes and *E* is the set of edges representing links. A layer is defined as graph *G′=(V,E′)* where *E′* is a subset of *E* and *G′* is a connected graph, i.e. a *layer* has only a subset of links in the network but contains all the nodes. A node is said to be safe in a layer if only one of its links is contained in that layer. The term safe layer for a node is used to denote a layer in which the node is safe. Many nodes can be safe in a single layer, where all such nodes are called safe nodes of that layer.

Fig. 3(a) shows an example network with eight nodes and fourteen links and it has no articulation points. Two example layers where all the nodes of the graph are safe are shown in Fig. 3(b) and 3(c).

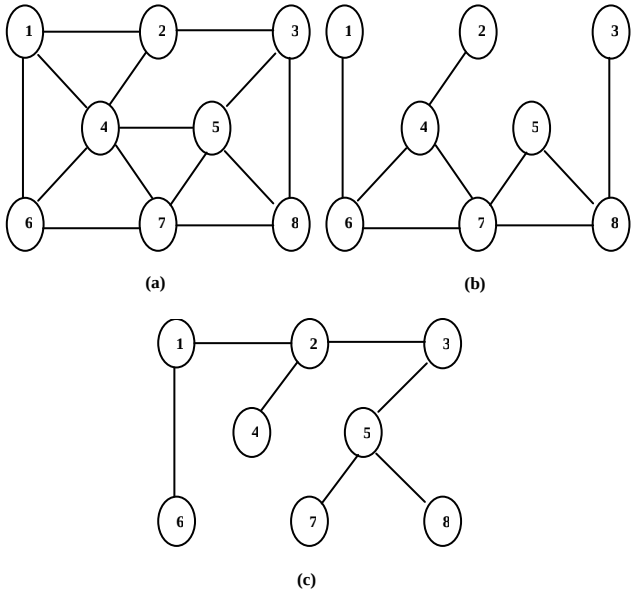

Fig. 3. Illustration of layers. (a) An Example Network with 8 nodes and 14 links. (b) Layer 1 generated based on (a). (c) Layer 2 generated based on (a).

B.2 Recovery Mechanism: In order to use an RRL as a complete method for recovery, layers are to be generated in such a way that all nodes that are not articulation points are safe in at least one layer. The layers are used as input to routing algorithms and a routing table is calculated for each layer. Tables containing routing information for each layer must be kept in every node.

When any node fails, all the traffic which is passing through the failed node is transmitted along the safe layer of that node. Only the traffic sourced by or destined for the failed node will be lost. Each packet is marked according to what layer is currently valid so that other nodes can keep forwarding in the same layer. All the packets not affected by fault are routed based on the full topology. The most important requirement for this scheme to work is detection and propagation of failure of a node so that the intermediate node can move the traffic to another layer.

C. Algorithms for computing QoS paths with restoration

This paper [5] gives algorithms for finding restoration topologies for QoS paths. A restoration topology is a set of bridges, each bridge protecting a portion of the primary QoS path. The authors claim that this approach is guaranteed to find a restoration topology with low cost when one exists. To choose a set of protective bridges that have minimum cost is a NP-hard problem. Hence this scheme has pseudo-polynomial time complexity.

The network has a source S that communicates with a destination D and has certain QoS requirements. The available bandwidth and delay on every link is known. It is assumed that the network does not contain any parallel links (i.e. two or more links that connect the same nodes). Only single failures of nodes or links are considered. Here the problem is to find a minimum cost restoration topology R for a given primary QoS path P.

The general approach is to construct an auxiliary graph which is a modified form of the original graph. Then the concepts like adjusted delay and feasible walk are used in the algorithms to construct restoration topology.

III. PROPOSED MODEL

It is understood that having larger number of backup segments enables one to capture the properties like faster recovery and better reuse of resources reserved along the primary path. Towards this end, the attention has been focused on improving the number of backup segments in the computed segmented backup.

The observation is that the numbers of backup segments computed by the *Min_SegBak* algorithm are less and in most of the cases it is an end-to-end backup even though there are possible backup segments. It has been demonstrated that proposed scheme produces more number of backup segments than *Min_SegBak* algorithm. A more detailed explanation of how the proposed scheme works is given in the following sections.

A. Overview

In the proposed scheme the idea is to divide the primary path into segments of equal size. Then construct auxiliary graph for each segment in such a way that the components of each segment are protected and the shortest path from source to destination in the graph has maximum overlap with the primary path. By making use of shortest path, the backup segment for the corresponding segment is segmented. Let 's' be the size of the segment. First, we choose a value for s to partition the primary path into segments in such a way that adjacent segments overlap with one link. This overlap ensures that recovery from node failures is guaranteed. For each segment a backup segment is produced by performing the following steps.

- Generate an auxiliary graph by removing the links that are incident on the intermediate nodes of the segment and make the remaining links along the primary path to have weight as zero.
- Find the shortest path P' from source to destination in the modified graph.
- Using primary path and P', compute the backup segment for that segment.

A smaller value for the segment size produces more backup segments which are shorter in length. Shorter backup segments take less time for failure notification and backup activation. In other words failure recovery is handled more locally. On the other hand larger values of segment sizes produce less number of backup segments which are longer in length. In case of long backup segments, time taken for failure notification and backup activation is relatively more.

The inference is that segment size is directly proportional to recovery time. However a smaller value for segment size incurs computational overhead. In our scheme, there is a flexibility of choosing the value of the segment size appropriately according to the delay requirement of the real-time application. In the next subsection the detailed illustration of the proposed scheme is given.

B. Illustration with Example

The section illustrates how the proposed method computes the segmented backup. Fig 4(a) shows the initial topology along with the primary path from source (*S*) to destination (*D*). In this example the segment size is assumed as three. Then partition the primary path into segments of size three with one link overlap as shown in Fig. 4(b) and compute auxiliary graph for the first segment by removing the links that are incident on the intermediate nodes of the segment and assign zero weight to rest of the links along the primary path as shown in Fig. 4(c).

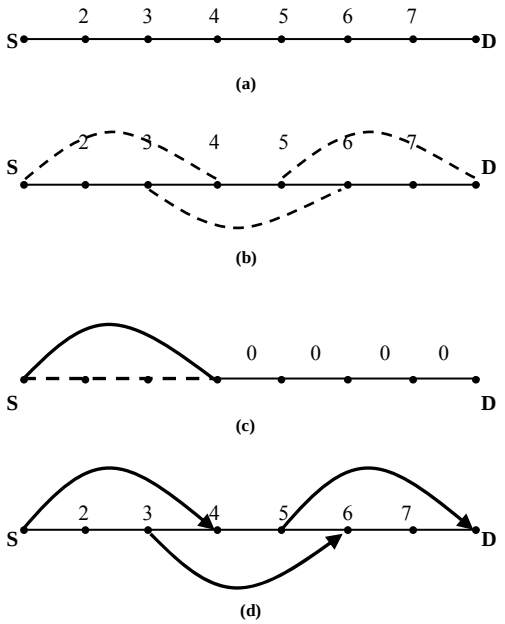

Fig. 4. Illustration with Example. (a) Initial topology with primary path from S to D. (b) Primary segments after segmentation of primary path. (c) Shortest path from S to D in the auxiliary graph of first segment. (d) Segmented backup consisting three backup segments.

Next find the shortest path P' from *S* to *D* in the auxiliary graph. By using primary path and P' compute backup segment for first segment. Similarly repeat the same procedure for all the segments to get a segmented backup as shown in Fig. 4(d).

The significance of the auxiliary graph is that it allows us to find a shortest path P' from S to *D* which will just bypass the intermediate nodes and links of the segment and reuse rest of the primary path. The zero weights to the links in the

modified graph force the backup path P' to reuse the primary path. The next section discusses the optimization issues of the proposed scheme.

C. Optimization

Here the scenarios are described, which show the need for improvement in the above said scheme. Fig. 5 shows instances of primary paths and their corresponding segmented backups, computed by our scheme. These instances are from a 12x12 mesh topology with random link weights ranging from 1 to 5. Fig. 5(a) shows an instance of a primary path and its segmented backup computed by our scheme. Backup segment (2) protects all the components that are being protected by backup segment (3). Hence, backup segment (3) is a redundant one. Fig. 5(b) shows another instance of primary path and segmented backup computed by our scheme. Here, five backup segments are constructed. One can observe that the backup segment (4) is protecting the nodes that are already protected by backup segments (3) and (5), so the backup segment (4) is redundant one.

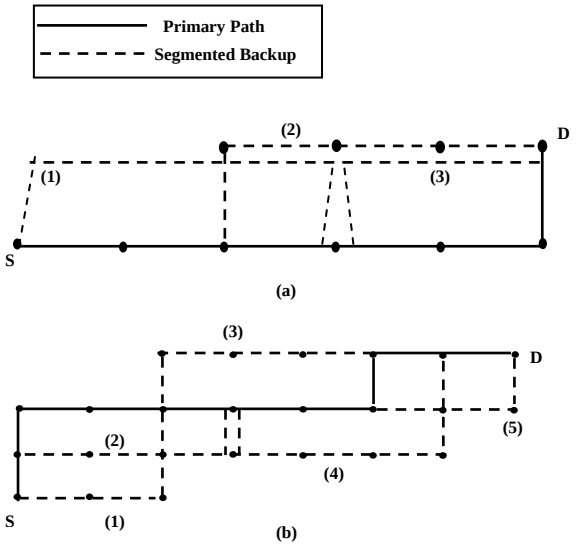

Fig.5 Segmented backups before optimization. (a) Example1 (b) Example 2.

In the process of our work the observation is that this redundancy is due to fact that there was no attempt to check if two segments share a backup segment leading to redundant backup segments. This is due to the reason that we partition the primary path a priori into segments of equal size. Consequently, the flexibility of choosing segments that are not redundant is loosed.

The proposed scheme is refined by making a small modification. We defer the computation of next segment until the current backup segment is computed. Once a backup segment is computed, start the next segment from the predecessor of the last node of the current backup segment.

Fig.6 shows how the modified scheme computes segmented backup for the same instances that are shown in the Fig. 5. Fig. 6(a) shows segmented backup computed by refined scheme for the instance shown in the Fig. 5(a). Here the segmented backup has only two backup segments and the computation of backup segment (3) is eliminated. Fig. 6(b) shows segmented backup computed by the refined scheme for the instance shown in Fig. 5(b).

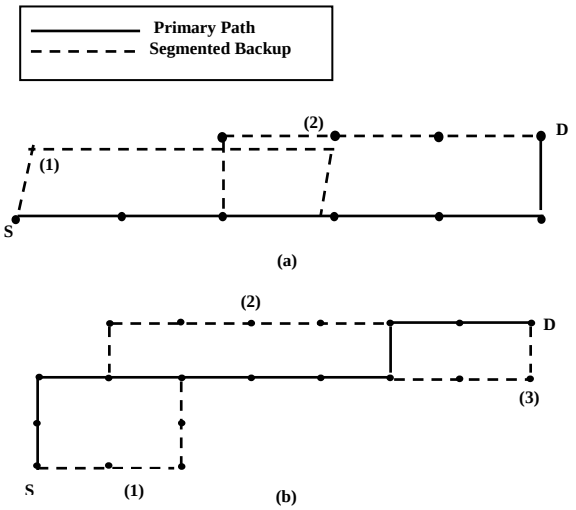

Fig. 6. Segmented backups after optimization. (a)Example 1. (b) Example 2.

In the earlier case, there are five backup segments. We can observe that the backup segment (4) is redundant since it is protecting the nodes that are already protected by backup segments (3) and (5). Even the backup segment (4) is removed; the segmented backup in the Fig. 5(b) consists of four backup segments whereas it is only three for segmented backup computed by the optimized scheme as shown in the Fig. 6(b).

Our optimized scheme works particularly well for arbitrary topologies where the backup segments tend to end at a node beyond the defined length of the segment.

D. Algorithms

Table I
NOTATION USED IN THE ALGORITHMS

$G(V,E)$	A graph that represents a network topology
V	Set of nodes of G
E	Set of links of G
S	Source
D	Destination
$P = (S,v1,v2,v3,.....,D)$	Sequence of vertices denoting the primary path in G
$PS = (ps1,ps2,.....psk)$	Sequence of segments of P
$G_i'(V,E_i')$	Auxiliary graph for segment ps_i
P_i'	Sequence of vertices denoting the shortest path between S and D in G_i'
$BS = (bs1,bs2.....bsk)$	Sequence of backup segments
s	Size of segment

Table I describes the notation used in the algorithms presented in Figures 7 to 9. The network is represented by a weighted graph $G(V,E)$, where V is the set of nodes and E is the set of links in the network. Edge weights in the graph

represent propagation delay of links in the network topology. Let *S* and *D* represent source and destination nodes respectively. The notation used is shown in Table I. In the notation, s represents number of *links* that a segment spans and bs_i represents the backup segment of the segment ps_i for $1 \le i \le k$.Where *k* is the number of backup segments in the segmented backup. The pseudo code of *Eff_SegBak(G,S,D,P,s)* is given in Figure 7.

```
Eff_SegBak(G,S,D,P,s)
 start_node set to Source S
 i set to 1
 loop
    Call  CAGraph(G,S,D, P, s, start_node) to construct auxiliary
          Graph and set to G_i'
    Call StPath(G_i',S,D) to compute shotest path P_i'
    Call CBSegment(P, P_i' ,S,D) to compute backupsegment  and set
          to bs_i
    Call getLastNode() for backup segment bs_i and set to start_node.
     if start_node is equal to destination D then
       end of loop
     end if
    Get the previous node on the primary path and set to start_node.
     increment i by one
 end loop
```

Fig. 7. Efficient segmented backup algorithm

The module $StPath(G_i',S,D)$ is any shortest path algorithm which finds shortest path between *S(source)* and *D(destination)* in the graph G_i'. In our implementation *Dijkstra's* algorithm is used for finding the shortest path. The module *CAGraph(G,S,D,P,s,start_node)* is used for construction of the auxiliary graph where start node represents the starting node of the segment for which the auxiliary graph is constructed and the algorithm is given in Figure 8.

```
CAGraph(G,S,D,P,s,start node)
 Set graph  G to G_i'
 Set count to one
 Set curr_node to source S
          //set the weights of all links along the primary path to zero
 while curr_node is not equal destination D do
   Get the next node on primary path and set Next_Node
   set the link delay between curr_node and next_node to zero in   G_i'
   Set the curr_node to  next_node
 end while
          //make the segment safe by setting the weights of links
          //incident on intermediate nodes of the segment to ∞
 Get the next_node on primary path and set to curr_node.
 while count is less than  segment size s and curr_node is not equal to
       destination D do
   for every link incident on curr_node set the link delay to ∞ in   G_i'
   Get the next node on  primary path and set to curr_node
   increment count by one.
 end while
 return G_i'.
```

Fig. 8. Auxiliary graph construction algorithm

First make a copy of the original graph G as G_i'. Then modify the graph G_i' to get the auxiliary graph of the segment ps_i. In the first while loop, make all links along the primary path P in the graph G to have weight as zero. In the second while loop; set weights of all links incident on the intermediate nodes of the segment to ∞.

The module $CBSegment(P,G_i',S,D)$ computes the backup segment for segment ps_i. Here *P* is the primary path in the graph *G* and G_i' is the shortest path from *S* to *D* in the auxiliary graph G_i' and the algorithm for the same is given in Figure 9.

```
CBSegment(P,P_i',S,D)
 Set first_found to false
 Set curr_node to  S
 while curr_node is not equal to destination  D do
   Get the next_node in primary path of Auxiliary graph and set
        to Next_Node
   if next_node belongs to  P and first_found is equal to false
                then
     Set curr_node to  next_node
   continue
   end if
  Add curr_node at the end of backup segment bs_i
  if next_node  doesn't belongs to P and first_found  is equal to
        false then
  Set to first_found to  true
     else if next_node belongs to P and first_found is equal to
        true then
    Add next_node at the end of backup segment bs_i
  out of while loop
  end if
  Set curr_node to  next_node
  end while
  return backup segment bs_i
```

Fig. 9. Backup segment computation algorithm

IV. EXPERIMENTAL RESULTS

This section evaluates the algorithms *Min_SegBack* and *Eff_SegBak* and explains the topologies that are used for testing and other implementation choices made by us. Later how these two schemes work with respect to the evaluation parameters is discussed.

A. Experimental Topologies

In our implementation the objective is to see how the algorithms *Eff_SegBak* and *Min_SegBak* compute backup segments. The behavior of the two algorithms tested on regular mesh topologies like 5 x 5, 7 x 7, 9 x 9 and 12 x 12. and other synthetic topologies generated by a tool called BRITE [3]. Two variants of BRITE generated topologies are used. The first is Router Waxman [3] and the second is Router Barabasi Albert [3].

The random delays for links in the topologies are ranging from 1 to 5. A delay constraint [5] is also imposed in our implementation according to which, if the total backup path length of a backup segment is greater than the predefined delay, that backup segment is not considered. The segment size chosen is three. The choice of segment size has greater impact on number of backup segments produced in *Eff_SegBak* and consequently affects the recovery delay. In the implementation, our primary concern is to see which of these two algorithms is better in producing larger number of backup segments. The implementation of our scheme is done in C++. Five hundred source-destination pairs are randomly generated for the above said topologies and computed these four parameters for each of such pairs using both the schemes and averaged the outputs of these five hundred trials.

B. Comparison of Number of Backup Segments

For a set of five hundred random source and destination pairs in a 12x12 mesh topology, number of backup segments is averaged by varying the segment size from 2 to 8, the result of which is depicted in the Fig. 10. It is clear from the graph that number of backup segments is inversely proportional to segment size. This is due to the fact that shorter segment size divides the primary path into large number of segments resulting in more backup segments.

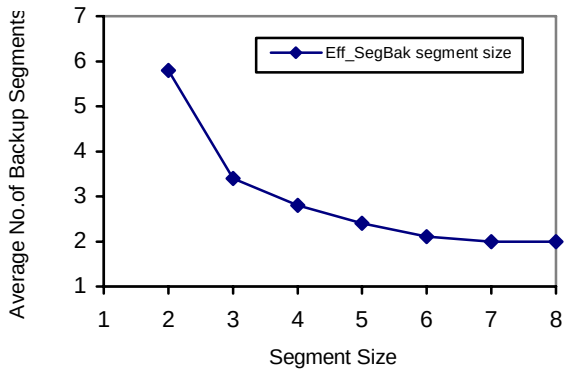

Fig. 10. Variation in number of backup segments with segment size with respect to *Eff_SegBack* algorithm.

The other factor which affects number of backup segments is the length of primary path. The graph in the Figure 11 shows the variation in number of backup segments with length of primary path.

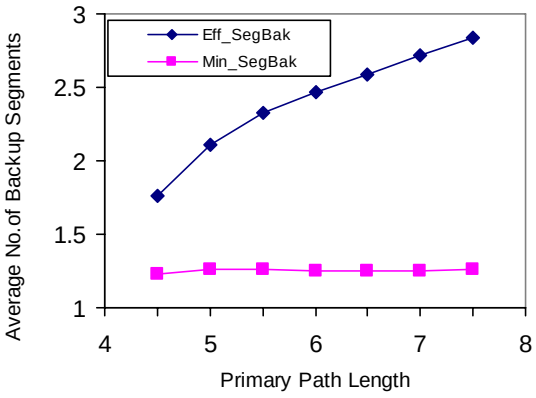

Fig. 11. Variation in number of backup segments with primary path Length

From the graph it is clear that the number of backup segments increases as the primary path length increases in case of *Eff_SegBak* algorithm, where as it is not the case with *Min_SegBak* algorithm.Since the number of backup segments increases with primary path length in case of *Eff_SegBak* algorithm, the length of backup segments is constant and the property of local recovery holds even for larger primary paths. From Table II the observation is that, *Eff_SegBak* gave more backup segments (ranging from 1.64 to 2.84) than that of *Min_SegBak* (ranging from 1.018 to 1.263) across all the topologies. Another observation is that *Min_SegBak* algorithm very rarely generates backup segments more than two. Note that figures shown are averaged for 500 samples of random source-destination pairs.

The proportionality of number of backup segments with primary path length is captured in our scheme, which is not the case with *Min_SegBak* algorithm. As the number of backup segments does not increase with primary path length in case of *Min_SegBak* algorithm, the backup segments tend to be larger in length for larger primary paths. Consequently the property of local recovery diminishes as the length of the primary path increases.

Table II
Number of backup segments for different topologies

Topology	5x5	7x7	9x9	12x12	Router_ Wax	Router_ BA
pp_len	4.20	5.43	6.65	7.66	4.71	4.44
Eff_SegBak	1.64	2.17	2.68	2.84	1.73	1.72
Min_SegBak	1.23	1.263	1.247	1.261	1.018	1.09

C. *Comparison of Percentage Reuse of Primary Path*

Min_SegBak algorithm gives percentage reuse ranging from 0.65 to 11.91 as given in Table III across all topologies under consideration. In case of *Eff_SegBak* it is ranging from 20.94 to 48.39. Reuse is much less in case of *Min_SegBak* because for most of the cases it gives end-to-end backup in which case the percentage reuse of primary path is zero.

Table III
Percentage Reuse of Primary Path for different Topologies

Topology	5x5	7x7	9x9	12x12	Router_ Wax	Router_ BA
Eff_SegBak	20.92	34.05	43.08	48.41	21.31	22.05
Min_SegBak	8.90	10.43	10.45	11.91	0.65	3.51

D. Average Hop Count for Failure Notification (AHFN)

From the Table IV the AHFN for *Eff_SegBak* is ranging from 1.60 to 1.84 which is less and consistent across all the topologies due to the fact that we are segmenting the primary path with a fixed segment size. In case of *Min_SegBak* algorithm, AHFN is more and varies from 1.92 to 3.43 with primary path length as shown in Fig. 12. This is due to the fact that the number of backup segments produced by *Min_SegBak* algorithm is not proportional to

Table IV
Average Hop count for Failure Notification of different Topologies

Topology	5x5	7x7	9x9	12x12	Router_ Wax	Router_ BA
pp_len	4.20	5.43	6.65	7.66	4.71	4.44
Eff_SegBak	1.60	1.69	1.75	1.84	1.84	1.72
Min_SegBak	1.92	2.47	3.0	3.43	2.34	2.14

the primary path length as much as in *Eff_SegBak*, resulting in variation of AHFN with length of primary path. Thus in case of *Min_SegBak* local recovery diminishes as primarypath length increases. With slighter AHFN the algorithm *Eff_SegBak* contributes significantly to faster failure recovery.

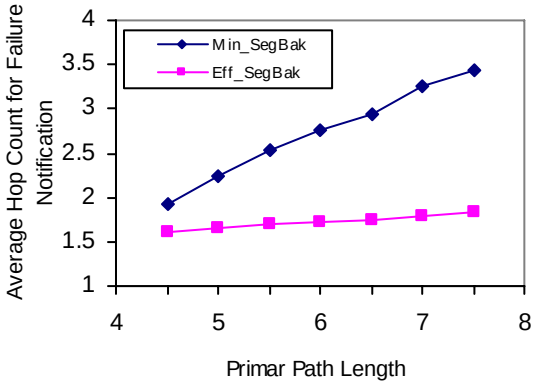

Fig. 12. Variation in AHFN with primary path

E. Comparison of Average Backup Path Length (ABPL)

From the Table V both the schemes are comparable with respect to ABPL. In Theorem 2 of [1], it is claimed that the segmented backup generated by the *Min_SegBak* algorithm is a minimum cost segmented backup for any chosen primary path. Since both the algorithms gave similar results with respect to ABPL, segmented backup produced by *Eff_SegBak* does not deviate much from the least cost one.

Table V
Average Backup Path Length for different Topologies

Topology	5x5	7x7	9x9	12x12	Router_ Wax	Router_ BA
Eff_SegBak	4.87	6.11	7.28	8.30	5.89	5.39
Min_SegBak	4.69	5.99	7.34	8.3	5.62	5.2

F. Exception Cases

We have tried to find cases where our scheme fails to give any backup segments and *Min_SegBak* gives end-to-end backup. In this section, such exception cases have found in the 12x12 mesh topology with random link weights ranging from 1 to 5 are described. We have labeled the nodes of the topology from 0 to 143. These cases are found very rarely.

Table VI shows instances of cases where our scheme fails to give any backup segments and *Min_SegBak* gives end-to-end backup that satisfies the delay constraint as shown.

Table VI
Exception cases with lesser values of delay constraint

Source	Destination	Delay Constraint	*Eff_SegBak*	*Min_SegBak*
109	65	25	0	end-to-end
80	124	20	0	end-to-end
86	47	15	0	end-to-end

When the delay constraint is relaxed to 30, the results for the same instances are shown in the Table VII. By relaxing the delay constraint *Eff_SegBak* could find backup segments covering all the components of the primary path.

Table VII
Exception cases where delay constraint is relaxed

Source	Destination	Delay Constraint	*Eff_SegBak*	*Min_SegBak*
109	65	30	4	end-to-end
80	124	30	3	end-to-end
86	47	30	4	end-to-end

V. CONCLUSIONS AND FUTURE WORK

In this paper, a new method has been presented for computing segmented backup for recovering from node and link failures in real-time IP communications. Our scheme gives more number of backup segments than the existing scheme *Min_SegBak* Algorithm. Consequently, it gives better results in terms of faster failure recovery and efficient reuse of primary path. Our scheme permits choosing appropriate value of segment size to satisfy the delay requirement of real-time applications. The proposed scheme gives consistent results for both mesh and random topologies. The future enhancements of our work include testing with more realistic data and optimization on choice of segment size versus resource utilization.

REFERENCES

[1] M. J. P. Krishna Phani Gummadi and C. S. R. Murthy, "An Efficient Primary-Segmented Backup Scheme for Dependable Real-Time Communications in Multihop Networks," *IEEE/ACM Transactions on Networking*, vol. 11, pp. 81–94, 2003.

[2] T. S. G. Audun Fosselie Hansen, Amund Kvalbein and O. Lysne, "Resilient Routing Layers for Recovery in Packet Networks," in *International Conference on Dependable Systems and Networks*, pp. 238–247, 2005.

[3] I. M. Alberto Medina, Anukool Lakhina and J.Byers, "BRITE: Universal Topology Generation from a Users Perspective," tech. rep., Boston University, 2001.

[4] B. M. Smita Rai and D. O. Deshpande, "IP Resilience within an Autonomous System: Current Approaches, Challenges, and Future Directions," *IEEE Communications*, pp. 142–149, October 2005.

[5] A. O. R. R. Yigal Bejerano, Yuri Breitbart and A. Sprintson, "Algorithms for Computing QoS Paths with Restoration," *IEEE/ACM Transactions on Networking*, vol. 13, pp. 648–661, 2005.

[6] S. D. J. Anderson, B. Doshi and P. Harshavardhana, "Fast Restoration of ATM Networks," *IEEE J. Select. Areas Commun*, vol. 12, pp. 128 – 139, January 1994.

[7] W Grover, "The Self-healing Network: A Fast Distributed Restoration Technique for Networks Using Digital Crossconnect Machines.," *IEEE GLOBECOM*, pp. 1090 – 1095, 1987.

[8] W. D. Grover and D. Stamatelakis, "Cycle-oriented Distributed Preconfiguration: Ring-like Speed with Mesh-like capacity for Self-planning Network Restoration," vol. 1, pp. 537–543, June 1998.

[9] S G F M Medard and R. A. Barry, "Redundant Trees for Preplanned Recovery in Arbitrary Vertex-redundant or Edge-redundant Graphs," *IEEE/ACM Transactions on Networking*, pp. 641–652, 1999.

[10] R. Kumar and S. Grace, "Fault-tolerant Model for Deterministic IP Communication," tech. rep., University of Hyderabad, 2006.

[11] A. Barabasi and R. Albert, "Emergence of Scaling in Random Networks," *Science*, pp. 509–512, October 1999.

Mr B Purushotham received B Tech(CSE) from J N T University, Hyderabad, India in 2005 and M Tech (CSE) from S V University College of Engineering, Tirupati in the year 2007. Currently he is working as Assistant Professor, Dept of Computer Science and Engg., C R Engineering College, Renigunta Road, Tiupati, India. His areas of interests are Computer Networks, Distributed Systems and Software Engineering.

Dr Ch D V Subba Rao received the B Tech (CSE) from S V University College of Engineering, Tirupati, India in the year 1991, M.E. (CSE) from M K University, Madurai in 1998 and Ph D (CSE) from S V University, Tirupati in 2008. He has 17 years of teaching experience. At present, he is working as Head, Dept of Computer Science and Engineering, S V University College of Engineering, Tirupati, India. His areas of interests include Distributed Systems, Operating Systems, Computer Architecture and Grid Computing. He is a member of IETE, IAENG, CSI and ISTE. He chaired and served as reviewer of number of international conferences viz. IASTED and IAENG. He visited Austria, Netherlands, Germany, Belgium and Hong-Kong.

Mrs Nimmagadda Padmaja received B E (ECE) from University of Mumbai, India in 1997 and M Tech (Communication Systems) from S V University College of Engineering, Tirupati in 2003. She has been pursuing her Ph.D. (Part-time) from S V University, Tirupati since 2006. Currently she is working as Associate Professor, Dept of Electronics and Communication Engg., C R Engineering College, Renigunta Road, Tiupati, India. Her areas of interests include Signal Processing, Communication Systems and Computer Communication Networks.