An Efficient Primary-Segmented Backup Scheme for Dependable Real-Time Communication in Multihop Networks

Krishna Phani Gummadi, Madhavarapu Jnana Pradeep, and C. Siva Ram Murthy, *Senior Member, IEEE*

***Abstract*—Several distributed real-time applications (e.g., medical imaging, air traffic control, and video conferencing) demand hard guarantees on the message delivery latency and the recovery delay from component failures. As these demands cannot be met in traditional datagram services, special schemes have been proposed to provide timely recovery for real-time communications in multihop networks. These schemes reserve additional network resources (spare resources) *a priori* along a backup channel that is disjoint with the primary. Upon a failure in the primary channel, its backup is activated, making the real-time connection dependable. In this paper, we propose a new method of providing backups called segmented backups, in which backup paths are provided for partial segments of the primary path rather than for its entire length, as is done in the existing schemes. We show that our method offers: 1) improved network resource utilization; 2) higher average call acceptance rate; 3) better quality-of-service guarantees on propagation delays and failure-recovery times; and 4) increased flexibility to control the level of fault tolerance of each connection separately. We provide an algorithm for routing the segmented backups and prove its optimality with respect to spare resource reservation. We detail necessary extensions to resource reservation protocol (RSVP) to support our scheme and argue that they increase the implementation complexity of RSVP minimally. Our simulation studies on various network topologies demonstrate that spare resource aggregation methods such as backup multiplexing are more effective when applied to our scheme than to earlier schemes.**

***Index Terms*—Backup channel, backup multiplexing, dependable connection, multihop network, primary channel, quality-of-service (QoS), real-time communication, resource reservation protocol (RSVP), segmented backup.**

I. Introduction

THE ADVENT of high-speed networking has introduced opportunities for new applications such as real-time distributed computation, remote control systems, digital continuous media (audio and motion video), video conferencing, medical imaging, and scientific visualization. Such distributed real-time applications demand quality-of-service (QoS) guarantees on timeliness of message delivery and failure-recovery delay. These guarantees are agreed upon before setting up the communication channel and must be met even in the case of bursty network traffic, hardware failure (router and switch crashes, physical cable cuts, etc.), or software bugs. Applications using traditional best effort datagram services like IP experience varying delays due to varying queue sizes and packet drops at the routers. To ensure bounded message delays for real-time applications, special schemes such as resource reservation protocol (RSVP) [15] have been proposed. In RSVP, resources (such as link bandwidths and router buffers) are reserved *a priori* along the message transmission path from the source to the destination for the duration of a *session*. While RSVP can provide QoS guarantees on the packet transmission latency, it lacks quick failure-recovery mechanisms. In RSVP, when a channel fails, a new one is established. A successful recovery cannot be guaranteed as sufficient resources might be lacking at recovery time. Further, the channel re-establishment time could take a long time, especially when there is contention for resources among disrupted channels. Given that some of these applications (such as commercial video on demand) last for a long time, ranging from several minutes to hours, such failures might not be uncommon during a single session.

The communications schemes designed to tolerate faults can be broadly divided into *proactive* and *reactive* schemes. In the former, the failure-recovery process runs throughout the duration of the message transmission in anticipation of failures, while in the latter, the recovery process is initiated after detecting failure. An example of a proactive scheme is the forward recovery or forward error correction scheme [3], [10], [20], in which multiple redundant copies of a message are sent along disjoint paths. This scheme has a huge resource overhead and is less desirable than lightweight reactive schemes, when infrequent packet losses are tolerable. In a simple reactive scheme, resources are reserved *a priori* along a path, called the *backup* path (or channel) [2], [5], [11], [28], which is disjoint with the path along which messages are being transmitted, which we shall refer to as the *primary* path (or channel). The spare resources reserved for a backup channel are activated only when its primary channel fails, ensuring quick and guaranteed recovery. When not in use, these spare resources can be used for best effort and other nonreal-time traffic to achieve better resource utilization than proactive schemes.

Two different reactive schemes have been analyzed for the establishment of backup channels. In the first, spare resources

Manuscript received December 13, 2000; revised July 22, 2001; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor A. Orda. This work was supported by the Department of Science and Technology, New Delhi, India.

K. P. Gummadi was with the Indian Institute of Technology, Madras 600036, India. He is now with the Department of Computer Science and Engineering, University of Washington, Seattle, WA 98195 USA (e-mail: gummadi@cs.washington.edu).

M. J. Pradeep was with the Indian Institute of Technology, Madras 600036, India. He is now with Microsoft Corporation, Redmond, WA 98052 USA (e-mail: pradeepm@microsoft.com).

C. S. R. Murthy is with the Department of Computer Science and Engineering, Indian Institute of Technology, Madras 600036, India (e-mail: murthy@iitm.ernet.in).

Digital Object Identifier 10.1109/TNET.2002.808405

in the vicinity of the failed component are used to reroute the channel. This method of *local detouring* [5], [28] leads to inefficient resource utilization, as after recovery the path lengths usually get extended significantly. In the second scheme, *end-to-end detouring* [2], [11] was proposed through the use of an end-to-end backup channel, i.e., a backup channel that extends from the source to the destination.

In this paper, we propose and evaluate a new scheme to construct backup paths, which we call *segmented backups*. A segmented backup comprises multiple backup paths, each spanning a contiguous portion of the primary path. This is unlike an end-to-end backup where the backup spans the entire length of the primary path. Using example scenarios and simulation studies, we show that segmented backups have numerous advantages over end-to-end backups.

- *Higher call acceptance rate*. This is due to primary paths that have a segmented backup but no end-to-end backup.
- *Improved network resource utilization*. This is because segmented backups are typically shorter than end-to-end backups and need less spare resources. Moreover, shorter backup paths lead to more efficient resource aggregation through *backup multiplexing* [4], [6], [8].
- *Better QoS guarantees*. A segmented backup can comprise multiple backups, each of which spans a part of the primary path rather than its full length. This allows for faster failure recovery and finer control of fault tolerance for long primary paths over components with varying reliability. Also, the backups could be chosen so that they result in minimal increases in end-to-end delays over primary paths.

It is possible to construct segmented backups optimized for different goals such as better resource utilization versus better failure-recovery delay. In this paper, we specifically provide algorithms for: 1) minimizing spare resources reserved by multiplexing segmented backups; and 2) constructing segmented backups that are optimal with respect to resource utilization.

The rest of the paper is organized as follows. In Section II, we explain the concept of segmented backups and illustrate their advantage over end-to-end backups using examples. In Section III, we give an algorithm for spare resource reservation and describe why our approach achieves better spare resource utilization than the existing methods. In Section IV, we present an algorithm for backup route selection and prove that it is optimal in the amount of spare resources reserved. In Section V, we extend RSVP with a failure-recovery procedure that is specific to our scheme and discuss the complexity of its implementation. We also explain scenarios in today's Internet where our scheme can be used. In Section VI, we evaluate the performance of our approach and demonstrate its superiority in terms of resource utilization (resource allocation efficiency) and call acceptance rate. We conclude the paper in Section VII.

II. Primary-Segmented Backup Scheme

In this section, we explain our segmented backup scheme [12] and its advantages over the end-to-end backup scheme. To establish dependable connections (fault tolerant and real-time connections), earlier schemes have used end-to-end backups, i.e., backups that run from the source to the destination without sharing any components with the primary path (other than the source and the destination nodes themselves). In our approach of *segmented backups*, we find backups for the primary path, taken in parts. The primary path is viewed as made up of smaller contiguous paths, which we call *primary segments*. We find a backup path for *each* segment, which we call *backup segment*, independently. By *segmented backup* we refer to these backup segments taken together. We illustrate these terms in Fig. 1 where a primary channel with intermediate nodes $N1$–$N8$ and links 1–9 is shown. The backup links are *A–K*. The primary channel has three primary segments, each with its own backup segment. The primary 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 primary segments of a primary path overlap on at least one link.

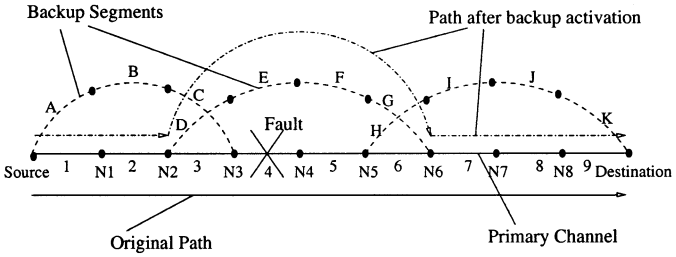

Fig. 1. Illustration of a primary channel with a segmented backup.

In Fig. 2(a), the goal is to establish a reliable connection from source node $N26(S1)$ to destination node $N5(D1)$. Suppose the primary channel is routed as shown in the figure, along one of the many shortest paths between them (typically, the primary path is chosen independently of the backup path). For this primary, it is impossible to find a backup path, while a segmented backup can be found as marked by the dotted line in Fig. 2(a). In fact, we can generalize the above observation into the following important theorem that guarantees an improved *call acceptance rate* (fraction of requested calls accepted at a given state of the network) for our scheme.

Theorem 1: In any network topology, whenever two disjoint paths exist between a pair of end nodes, backup segments are guaranteed to exist for any choice of a primary path between them. Similar guarantees cannot be provided on the existence of end-to-end backup (proof given in Appendix I).

The advantage of using segmented backups for more efficient resource utilization is illustrated in Fig. 2(b). There is a dependable connection to be established between $N19(S1)$ and $N11(D1)$. The primary path, end-to-end backup, and segmented backup are routed as shown. For simplicity, let us assume that both primary and backup paths need one unit of resource to be reserved per link. We can see that while the end-to-end backup requires eight units of resource, the segmented backup requires only seven units of resource (it has two backup segments needing three and four units). Recall our earlier observation that every end-to-end backup is also a segmented backup. If we could find the segmented backup that consumes minimum resources (referred to as *minimum segmented backup*), we are assured of better than or equal

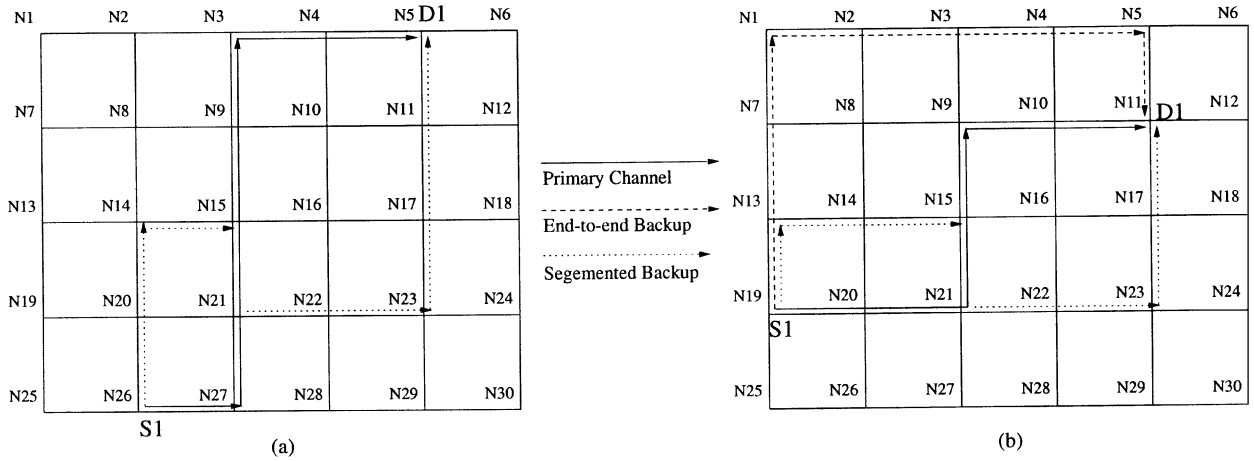

Fig. 2. Advantages of segmented backups over end-to-end backups. (a) *Higher call acceptance rate* for primary paths with a segmented backup but no end-to-end backup. (b) *Improved resource utilization and QoS guarantees* with segmented backups that are shorter than end-to-end backups.

resource consumption when compared with the end-to-end backup scheme. In Section IV, we present an efficient algorithm for finding the minimum segmented backup. Further, in Fig. 2(b), a failure of a node or a link in the six-hop-long primary channel results in a re-established path that is: 1) eight hops long using end-to-end backups; and 2) six hops long using segmented backups. The lesser hop counts translate to better QoS guarantees on delay.

Let us see why resource sharing algorithms such as backup multiplexing result in greater gains while using segmented backups. (The model is explained in detail in Section III. If unclear, we suggest revisiting this example after reading Section III.) Using backup multiplexing, backup resources reserved for primary channels that do not have any common components can be shared. A shorter primary channel shares components with fewer primary paths, leading to better multiplexing of its backup path. In the case of segmented backups, two backup segments can be multiplexed whenever their corresponding primary segments do not share any components. As primary segments are shorter than primary paths, backup segments tend to multiplex more with other backup segments than end-to-end backups, leading to greater gains in resource savings.

We illustrate the intuition through Fig. 3, where we try to establish two dependable connections: $N19(S1)$–$N5(D1)$ and $N25(S2)$–$N12(D2)$. Suppose their primary paths, end-to-end backups, and segmented backups are routed along the shortest paths, as shown. The primary paths share a single shared node $N11$. Assume that both primary and backup paths need one unit of resource to be reserved per link. Their end-to-end backups cannot be multiplexed on the links they share, i.e., links from $N5$–$N6$ and $N6$–$N12$. Hence, the total spare resources reserved equals 19 units (nine units for the first channel plus ten units for the second channel). In contrast, segmented backups need only twelve units of reserved resources. This is because the primary segment from $N19$–$N10$ on the first channel and the primary segment from $N25$–$N11$ on the second channel do not have any shared components. This allows their backup segments to be multiplexed on links from $N19$–$N10$. So, the total spare resources reserved equals twelve units (eight units for channel 1 plus nine units for channel 2 minus five units for the links on which backup segments are shared).

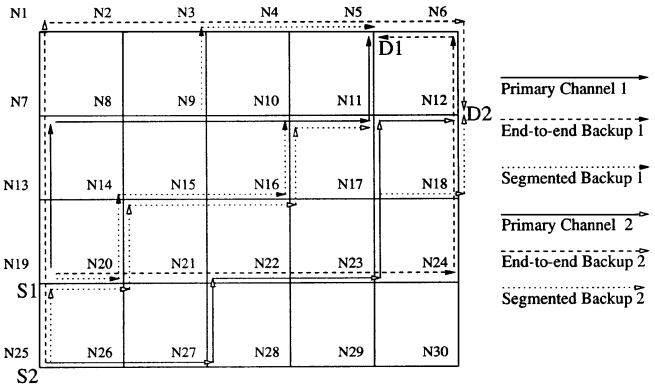

Fig. 3. Advantages of segmented backups over end-to-end backups: *improved resource aggregation* as backup multiplexing is more effective using segmented backups.

To summarize, we explained the concept of segmented backup scheme and pointed out ways in which it is more efficient over the end-to-end backup scheme. Perhaps the best thing with our scheme is that it encompasses the end-to-end backup scheme as well, and thus guarantees at least the performance of the end-to-end backup scheme in the worst case. Of course, this might require a few routers to store more state information and perform more processing. In the rest of this paper, we provide algorithms and describe methods to realize this concept and quantify its potential through simulation studies.

III. Efficient Spare Resource Allocation

The spare resources reserved lower the maximum throughput of the system as the resources reserved for backup channels are used only during component failures. So, minimizing spare resources is an important metric while evaluating different schemes. In this section, we briefly outline a method to minimize the amount of spare resources reserved by multiplexing (sharing) the backups passing through the same link. The technique of backup multiplexing is discussed in detail in [4], [6] and [8], and we adapt it here to our case of segmented backups.

We consider the *single-link* failure model for our analysis, under the assumption that component (link) failure-recovery time, i.e., time taken for the fault to be rectified, is much smaller

than the network's mean time to failure (MTTF). Using this failure model, if 1) primary channels of two connections share no common components and 2) their backup channels with bandwidths $b1$ and $b2$ pass through link L, it is sufficient to reserve $\max(b1, b2)$ for both the backup channels on the link L. This is because, in this model, both of the backup channels will never be activated simultaneously. This technique of sharing backup resources is known as backup multiplexing or, more specifically, *deterministic* backup multiplexing, as recovery is always guaranteed in this model even after multiplexing. We discuss a different model called *probabilistic* backup multiplexing later in this section.

An efficient algorithm to calculate the spare resources S_L at link L under the single-link failure model is given below. Let Φ_L denote the set of all primary segments whose backups traverse L. Let R_{P_S} denote the resource required at each link by the primary segment P_S.

```
Initialize S_{I,L} = 0, ∀ I, L
loop for each link I, I ≠ L
 loop for each primary segment P_S ∈ Φ_L
  if P_S contains link I then
   S_{I,L} = S_{I,L} + R_{P_S}
  endif
 endloop
endloop
S_L = max{S_{I,L}}
```

Note that to employ this algorithm, it is necessary for each router to be aware of the primary segments for all backup segments passing through it. This algorithm also provides some rationale for our earlier observation that backup segments tend to multiplex more than end-to-end backups. The primary segments in our scheme are shorter than the primary channels in the case of end-to-end backups. So, the condition for the **if** statement in the innermost loop holds true fewer times for segmented backups than for end-to-end backups, requiring fewer resources to be reserved. To obtain greater improvements in the backup segments' multiplexing capability over end-to-end backups' multiplexing capability, one needs longer primary paths with larger number of segments per backup. Such scenarios are more likely to occur as the network size increases. Hence, we expect our scheme to show increasing gains with increases in the size of the network. Our simulation studies reported in Section VI support this expectation.

Below, we briefly discuss how probabilistic backup multiplexing applies to our scheme. In this model, each network component (link or node) fails with a certain probability λ and two backup segments are multiplexed on a shared link only if the probability of their simultaneous activation is less than a threshold ν, called the multiplexing degree. For each link, let $\mathrm{Prob}(B_{i,k}, B_{j,l})$ denote the probability of simultaneous activation of two backup segments, $B_{i,k}$ (the kth segment of backup B_i) and $B_{j,l}$ (the lth segment of backup B_j). Let their corresponding primary segments be denoted by $P_{i,k}$ (the kth segment of primary P_i) and $P_{j,l}$ (the lth segment of primary P_j). Then, backups $B_{i,k}$ and $B_{j,l}$ can be multiplexed only if

$$\begin{aligned}\mathrm{Prob}(B_{i,k}, B_{j,l}) &= 1 - (\text{probability of no failure in shared components between } P_{i,k} \text{ and } P_{j,l}) \\ &\quad \times (\text{probability of no simultaneous failures in the rest of the unshared components}) \\ &\leq \nu.\end{aligned}$$

The smaller the value of ν, the higher is the fault tolerance for the connections. This multiplexing degree ν can also be made specific to each connection, with higher fault tolerance for more critical connections or higher paying customers. In our scheme, we can control fault tolerance at the granularity of the segment of a primary path. Such fine control is useful as segments of the same primary path can differ in their reliabilities (probably because they comprise a different number of components with varying reliabilities). Yet another way of providing higher fault tolerance for a connection is by establishing multiple backup paths. Often, a few links are more critical or error prone than others. It is useful to have multiple backups for a short primary segment spanning such links rather than for the entire path. A detailed study of the extra flexibility offered by segmented backups over end-to-end backups under probabilistic multiplexing model is provided in [19]. In this paper, we restrict our discussion to deterministic multiplexing.

IV. Backup Route Selection

It is usually desirable to select a segmented backup with minimum *backup delay increment* (i.e., the difference between delays along the primary and backup paths) or minimum cost. Elaborate routing methods that search for routes using various QoS metrics have been discussed in [14], [18], [19], [24], [26], and [27]. In [19], we deal with construction of segmented backups that offer better QoS guarantees on reliability. However, we restrict our discussion here to construction of segmented backups with minimum cost, where cost can be a function of path delay or path length or path resource reservation requirement. Though our goal is to design algorithms to select minimum cost segmented backups, it follows from Theorem 1 that such algorithms would also yield a higher call acceptance rate.

The problem of optimal routing of backups is NP-hard as it subsumes the following problem which is known to be NP-hard [26]: *Is there a feasible set of channel paths such that the sum of traffic flows at each link is smaller than the link capability, when traffic demands are given?* The complexity of the problem increases greatly if one considers backup multiplexing. So, we resort to heuristics to select least cost backups to each individual primary path, ignoring the additional savings offered by multiplexing. Several greedy heuristics for selecting end-to-end backup paths are discussed in [6]. A simple way to find a minimum-cost end-to-end backup for a primary path in a network graph G is to use some shortest path search algorithm such as Dijkstra's over a graph G' obtained from G by removing the the components along the primary path. The problem of selecting minimum-cost segmented backups is, however, more difficult as typically there are a larger number of segmented backups

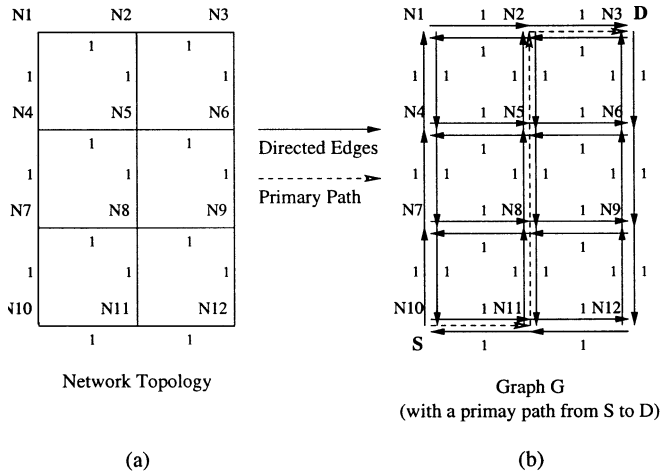

Fig. 4. Modeling a network topology as a graph. (a) 3×4 mesh topology with link weights. (b) Weighted directed graph G representing the network topology with a chosen primary path between two vertices S and D.

than end-to-end backups and we have to find intermediate nodes where the backup segments meet the primary. Below, we provide an algorithm called *Min_SegBak* that solves this problem.

Algorithm Min_SegBak: We model the network topology as a weighted directed graph $G(V, E)$. Every node n in the network is represented by a unique vertex v in the vertex set V, while every link l from node $n_1(v_1)$ to $n_2(v_2)$ with cost c is represented by a directed edge $e_1(v_1, v_2)$ from v_1 to v_2 with weight c. (A duplex link is replaced with two links in either direction with the same cost.) Let S and D denote the source and destination nodes, respectively, for a dependable connection and $P = S, i_1, i_2, \ldots, i_p, D$, a sequence of vertices along some primary path between them. We define $\mathrm{succ}_P(x, y)$ ($\mathrm{pred}_P(x, y)$) to denote that vertex x that occurs after (before) vertex y in sequence P. In Fig. 4(a) and (b), we show a 3×4 mesh topology with links of unit cost and the corresponding weighted directed graph.

Our algorithm to find the shortest segmented backup for P in G consists of three steps. In step 1, we generate a modified graph $G'(V, E')$ from $G(V, E)$ on the same set V of vertices. In the step 2, we find the shortest path between S and D in graph G' and use it in step 3 to obtain the segments of the minimum-cost segmented backup in G. We elaborate on this below and illustrate it in Fig. 5.

1) *Generate modified graph $G'(V, E')$ from $G(V, E)$.* The following types of edges in graph $G(V, E)$ are modified in the order given below:
   a) edges between successive vertices in the sequence P; the edges pointing in the direction from S to D are removed while those pointing in the reverse direction are assigned zero weight.
   b) edges $e(v_1, v_2) \in E, \ni v_2 \in P - \{S, D\}$; replace every such edge e with $e'(v_1, v_2')$ where v_2' is the immediate predecessor of v_2 in P. In other words, redirect every edge pointing to a vertex $v_2 \in P$, to point to its immediate predecessor vertex $v_2' \in P$.
2) *Find the shortest path between S and D in $G'(V, E')$.* Run any least cost path algorithm for directed graphs (e.g., Dijkstra's algorithm) between S and D in G'. Let the path obtained be denoted by the vertex sequence $B = S, i_1', i_2', \ldots, i_m', D$.
3) *Use the shortest path found in step 2 to find backup segments for P in G.* Suppose the segmented backup consists of backup segments $BS_1, BS_2, \ldots, BS_b$, ordered by the increasing position of their starting nodes in the sequence P. We describe a method to generate the vertex sequences for these backup segments one after the other (i.e., first BS_1 is generated, then BS_2, and so on until BS_b) as we traverse the sequence $B = S, i_1', i_2', \ldots, i_m', D$ (found in step 2).

Initialize all vertex sequences $BS_1, BS_2, \ldots, BS_b$ to empty. At any stage of the traversal of B, we use BS_c to denote the current backup segment being generated and i_c' to denote the current vertex. Initialize i_c' to S. For every i_c' perform the first applicable action indicated in steps a–d in that order. This process terminates on reaching destination D. We use the terms $\mathrm{prev}(i_c')$ and $\mathrm{next}(i_c')$ to denote the immediate predecessor and immediate successor vertices of i_c' in P. Similarly, $\mathrm{next}(BS_C)$ denotes the immediate successor of BS_C in B.

a) If $i_c' = S$ then $BS_c = BS_0$ and $i_c' = \mathrm{next}(i_c')$.
b) If $i_c' = D$ then
   i) If $\mathrm{prev}(i_c') \notin P$ then append $\{i_c'\}$ to BS_c and *stop*.
   ii) If $\mathrm{prev}(i_c') \in P$ then $BS_c = \mathrm{next}(BS_c)$, append $\{\mathrm{prev}(i_c'), i_c'\}$ to BS_c and *stop*.
c) If $i_c' \neq i_k$ for any $k \leq p$, (i.e., $i_c' \notin P$) then
   i) If $\mathrm{prev}(i_c') \notin P$ then append $\{i_c'\}$ to BS_c and $i_c' = \mathrm{next}(i_c')$.
   ii) If $\mathrm{prev}(i_c') \in P$ then $BS_c = \mathrm{next}(BS_c)$, append $\{\mathrm{prev}(i_c'), i_c'\}$ to BS_c and $i_c' = \mathrm{next}(i_c')$.
d) If $i_c' = i_k$ for some $k \leq p$, (i.e., $i_c' \in P$) then
   i) If $\mathrm{prev}(i_c') \notin P$ then append $\{i_{k+1}\}$ to BS_c and $i_c' = \mathrm{next}(i_c')$.
   ii) If $\mathrm{prev}(i_c') \in P$ and $\mathrm{pred}_p(\mathrm{prev}(i_c'), i_c')$ then $BS_c = \mathrm{next}(BS_c)$, append $\{\mathrm{prev}(i_c'), i_c'\}$ to BS_c and $i_c' = \mathrm{next}(i_c')$.
   iii) If $\mathrm{prev}(i_c') \in P$ and $\mathrm{succ}_p(\mathrm{prev}(i_c'), i_c')$ then $i_c' = \mathrm{next}(i_c')$.

The resulting vertex sequences define backup segments which form the shortest segmented backup for the primary path P in G.

We now illustrate the three steps above through an example in Fig. 5. The weighted directed graph G obtained from the 3×4 mesh topology is shown in Fig. 5(a), while the graph G' obtained by modifying the graph G using step 1 is shown in Fig. 5(b). Edges along the primary path in the direction from S to D (for example, the edge from $N5$ to $N2$) are removed, while those in the reverse direction (for example, the edge from $N2$ to $N5$) are assigned zero weight. Also, edges pointing to the nodes on the primary path (for example, the edge from $N4$ to $N5$) are redirected to point to preceding vertices on the primary path (the edge from $N4$ to $N5$ is redirected to $N8$). The shortest path in G' found in step 2 of the algorithm is also shown in Fig. 5(b) with a dotted line. Finally, a segmented backup consisting of two backup segments ($\{S, N7, N4, N1, N2\}$ and $\{N5, N6, D\}$) obtained from the shortest path using step 3 of the algorithm is shown in Fig. 5(c).

Note that in step 1b we redirect the edges to ensure that successive backup segments overlap on at least one link of the primary path. This overlap is necessary to recover from node fail-

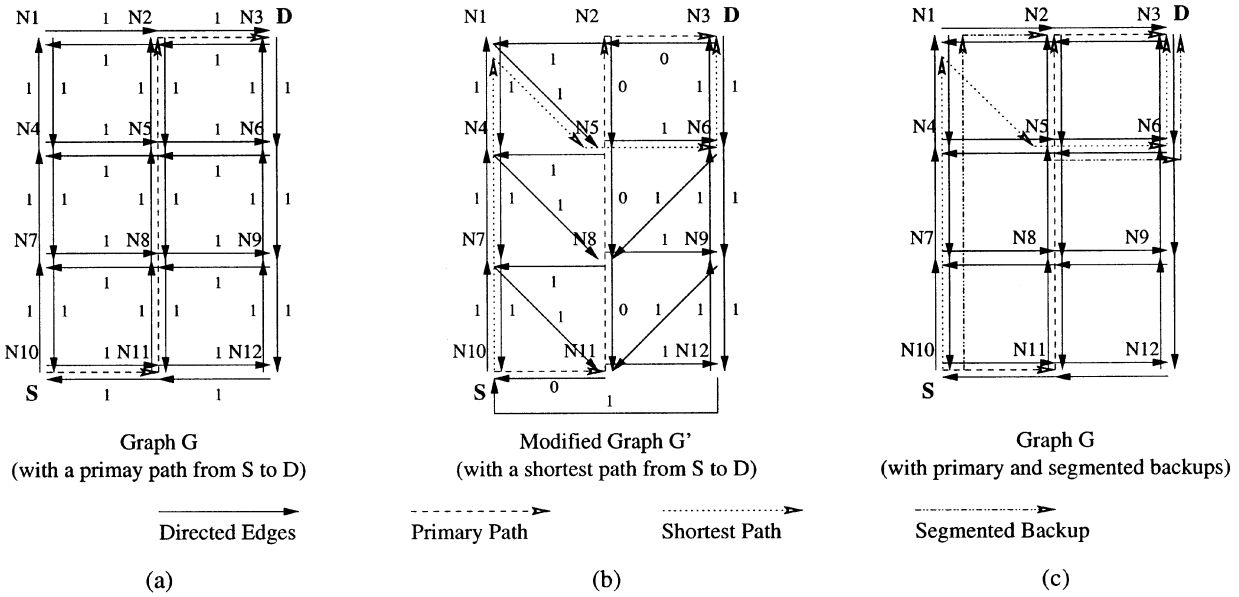

Fig. 5. Illustration of *Min_SegBak* algorithm. (a) Weighted directed graph G with a chosen primary path between two vertices S and D. (b) Modified graph G' obtained from G along with a shortest path between S and D; these are obtained using steps 1 and 2 of the algorithm. (c) Directed graph G with the primary and a segmented backup; the backup is obtained from the shortest path using step 3 of the algorithm.

ures. If, however, it is sufficient to recover from link failures and not necessarily node failures, step 1b can be omitted from the algorithm. We now state an important theorem of optimality of the segmented backups generated. This theorem, coupled with the fact that every end-to-end backup is also a segmented backup, guarantees a more efficient resource reservation using our algorithm.

Theorem 2: The segmented backup generated by the *Min_SegBak* algorithm is a minimum cost segmented backup for the chosen primary path (proof given in Appendix II).

Complexity of the Min_SegBak algorithm: The complexity of step 1 of the algorithm is at most the number of edges in $G(V, E)$, which is $O(|E|)$. Step 2 involves finding the least weight path in $G'(V, E')$; assuming one uses Dijkstra's algorithm, its complexity is $O(|V|^2 + |E'|)$. The complexity of step 3 is the cost of traversal of the shortest path in $G'(V, E')$ which is $O(|V|)$. Since $|E'| < |E|$, the overall complexity of the algorithm is $O(|V|^2 + |E|)$ which is the complexity of the least weight path algorithm over $G(V, E)$. This is same as the complexity of end-to-end backup. Thus, our algorithm offers improved resource utilization without additional complexity.

V. Implementation of Primary-Segmented Backup Scheme

In this section, we deal with the protocols required to implement our scheme, their complexity, and their applicability in the Internet as it exists today. We discuss three protocols: 1) *routing protocol*, to determine the primary as well as the backup paths; 2) *reservation protocol*, to reserve resources efficiently along these paths; and 3) *recovery protocol*, to recover from any failure. We present the recovery protocol as a simple extension to the Internet Engineering Task Force (IETF)-recommended resource reservation protocol, RSVP [15], and show that its complexity is comparable to that of RSVP. Finally, we reflect on the relevance of these schemes in the rapidly evolving Internet.

A. Design and Analysis of Implementation Protocols

In order to implement our scheme we need two protocols, namely, routing and reservation-recovery protocols.

Design of the Routing Protocol: The routing protocol determines the primary and segmented backup paths using the *Min_SegBak* algorithm described in Section IV. To compute the backup path, our algorithm assumes that every router has the global knowledge of the network topology. To obtain information about all routers and links in the network, our routing protocol can use any variant of link state protocol such as the Open Shortest Path First (OSPF) [17] protocol (a popular intra-domain routing protocol). With this global knowledge of the network, the primary path can be computed as the least weight path using Dijkstra's shortest path algorithm, while the backup path could be decided later by running the *Min_SegBak* algorithm at the source node.

Analysis of the Routing Protocol: We showed in Section IV that the complexity of the *Min_SegBak* algorithm is same as that of the complexity of the least weight path algorithm such as Dijkstra's algorithm, which is also the complexity of the widely used OSPF routing protocol [16]. Thus, while our protocol does require extra computations at routers for the backup paths compared with OSPF, the increased computation is bounded by a small constant factor. We feel that this should not be a problem in these days when Moore's law ensures that the computation power of the router processors doubles every two years.

Further, note that using link state protocols could potentially limit the scalability of our scheme to within a single autonomous system. While we believe that our subsequent work [21] on a distributed version of the *Min_SegBak* algorithm could eliminate this limitation (as it does not need global knowledge of any kind), we do not explore it here as it is outside of the scope of this paper. However, later in this section, we show that there are interesting applications even with this scalability bottleneck.

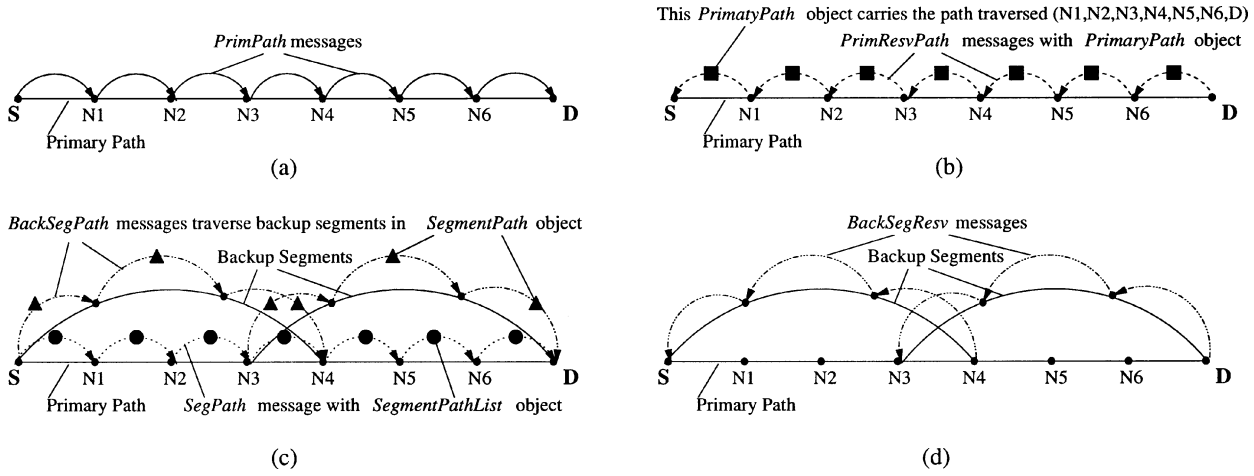

Fig. 6. Illustration of resource reservation using setup messages. (a) Source initiates the process by sending a *PrimPath* message. (b) Destination responds with a *PrimResv* message. (c) Source computes backup segments and sends *SegPath* message. As it traverses, the start nodes of backup segments initiate *BackSegPath* messages. (d) End nodes of backup reply with *BackSegResv* messages.

Design of the Reservation Protocol: We propose simple extensions to RSVP [15] to design the reservation protocol. These extensions are along the lines of extensions discussed in [4]. We begin with a brief overview of RSVP.

RSVP is a protocol to reserve resources along the network paths for unicast and multicast data flows in an integrated services network. The protocol primarily consists of the following three types of messages: 1) *setup messages* to reserve resources along paths; 2) *maintainance messages* to notify the end nodes of any failures along the path; and 3) *teardown messages* to free the resources reserved. We will discuss a few important messages of each type.

1) *Setup messages*. The source router initiates the reservation by sending a *Path* message to the destination along a route selected by the routing protocol. On receiving the message, the destination router sends a *Resv* message back to source in the reverse direction using the path state set up by the *Path* message. Resources are reserved at the routers along the path by the *Resv* message.
2) *Maintenance messages*. Error messages such as *PathErr* and *ResvErr* are used to inform the end nodes of a failure in establishing the path.
3) *Teardown messages*. Messages like *PathTear* and *ResvTear* are propagated much like their setup message counterparts to reclaim the resources reserved for the data flow.

In our reservation protocol, we have additional messages to reserve resources along the backup paths. We also add a few objects to the messages discussed above.

1) *Setup messages*. We illustrate the connection setup process in Fig. 6. The reservation is initiated by the source with a *PrimPath* message to the destination as shown in Fig. 6(a). The destination responds with a *PrimResv* message back to the source. The *PrimResv* message contains an object called *PrimaryPath*, which is filled up with the routers along the primary path as the message traverses it, as shown in Fig. 6(b). This *PrimaryPath* object is used by the source to compute the primary and backup segments using *Min_SegBak* algorithm. Each pair of primary and backup segments is copied into a separate *SegmentPath* object. All the *SegmentPath* objects are copied into a *SegmentPathList* object, which is sent in a *SegPath* message from the source to the destination. As the *SegPath* message traverses the primary path, the *SegmentPathList* is used by routers to identify the start nodes of backup segments. These nodes initiate a *BackSegPath* message along the backup segment. This message is tagged with the *SegmentPath* object (contains the corresponding primary and backup segments) to help in backup multiplexing. This is illustrated in Fig. 6(c). On receiving the *BackSegPath* message, the last node of a backup segment replies with a *BackSegResv* message, which traverses the backup segments in reverse direction as shown in Fig. 6(d). Resources are reserved along the backup segments by the *BackSegResv* message using backup multiplexing.
2) *Maintenance messages*. Upon failure during the set up of either primary path or any backup segment, error messages like *PrimPathErr*, *PrimResvErr*, *BackSegPathErr*, and *BackSegResvErr* are generated. These messages also initiate the corresponding reservation teardown messages to free any resource reservations made prior to the routing failure.
3) *Teardown messages*. The messages *PrimPathTear*, *PrimResvTear*, *BackSegPathTear*, and *BackSegResvTear* are used to free the resources reserved. They are propagated along the primary and backup paths.

Analysis of the Reservation Protocol: The reservation state maintained by the routers is soft state and must be updated periodically through refresh messages. If a periodic refresh message is not received, a router can reallocate the reserved resources for some other flow. Thus, the resources can be recovered even when the source or destination nodes fail to generate explicit *Teardown* messages.

The amount of state maintained at the routers by the reservation protocol is a very important concern for integrated services. Below, we argue that the state stored in our scheme is only marginally greater than the state maintained in the

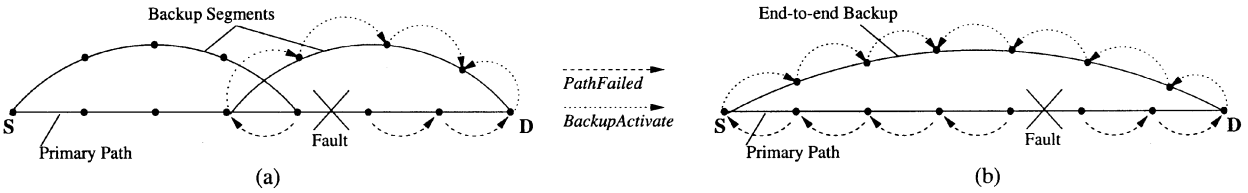

Fig. 7. Illustration of failure recovery using *PathFailed* and *BackupActivate* messages in (a) segmented backups and (b) end-to-end backups.

end-to-end backup schemes and is comparable to the state required by RSVP.

Both in the end-to-end backup scheme and in our scheme, all routers along the primary as well as the backup paths maintain per-flow state. However, our scheme demands that extra state be stored at a few intermediate routers responsible for initiating failure recovery (describe later in this section). This is because, unlike end-to-end backup schemes, where the recovery can be initiated only by the source or destination nodes, in our scheme, every router at which a backup segment is initiated or terminated (e.g., S, $N2$, $N3$, $N5$, $N6$, and D in Fig. 1) can initiate the recovery process. When the number of segments is b, there will be $2(b-1)$ routers that store additional information about the primary segment for which they are responsible. Assuming an average path length of 20[1] and an average primary segment length of 6 (with successive segments overlapping on one hop), the expected number of backup segments is four. This translates to six routers maintaining an additional state, which we believe is a small overhead for the additional benefits using our scheme.

In RSVP-based real-time communication, routers store reservation only for the primary path (note that RSVP does not offer failure recovery), while in our scheme as well as in end-to-end backup schemes, routers have to store state for both primary and backup paths. Typically, the length of a segmented backup is of the same order of magnitude as the primary path length. Thus, our scheme as well as the end-to-end backup scheme maintains about twice the amount of state as maintained by RSVP. We believe that this is a practical tradeoff for the failure recovery guarantees offered and that it would be feasible to deploy these schemes wherever RSVP is deployed. Further, if backup multiplexing were to be employed to reduce the spare resources reserved, routers along the backup segments need to maintain additional state about their correponding primary segments. We must acknowledge here that several researchers hold the opinion that RSVP in the integrated services framework might never be widely adopted given its demand for routers to maintain per-flow state. While we remain optimistic that increasing demand for applications such as video conferencing will eventually lead to adoption of schemes such as RSVP, we also explore alternate application scenarios in the current Internet later in this section.

Design of the Recovery Protocol: Failure recovery comprises three phases: detecting the fault, reporting the failure, and activating the backup.

In our model, we assume that when a link fails, its end nodes can detect the failure, and that when a node fails, all its neighbors can detect the failure. For failure detection techniques and their evaluation, refer to [7]. We do not discuss them further here. The nodes that detected the fault report it to the start and end nodes of the corresponding backup segment, which then activate the backup to recover from the failure. We now introduce a new type of message, called the *failure-recovery message*, to report failures and to activate the backup.

Failure-Recovery Messages: The routers that detect the failure send a *PathFailed* message toward the source router and destination routers along the primary path. The *PathFailed* message contains the router(s) where the failure occurred (if a link fails, both its neighboring routers are reported). This message propagates, tearing down the resourcers reserved along the primary path, until it reaches the routers at the start and the end of the primary segment containing the faulty component. These nodes start the activation of the backup segment by sending an *BackupActivate* message along the backup segment. Both the end nodes of a backup segment are involved for faster failure recovery. We illustrate this failure-recovery process in Fig. 7.

Analysis of the Recovery Protocol: An important metric to analyze the recovery protocol is failure-recovery delay, which is the time taken to re-establish the service. This delay also determines the number of lost messages and it is critical for many real-time applications to minimize it. We compare the recovery delay in our scheme with that in the end-to-end backup scheme.

In the end-to-end backup scheme, the failure reports, i.e., the *PathFailed* messages have to reach the source and destination before the backup activation begins, while in our scheme the backup activation starts as soon as the messages reach the end nodes of the primary segment containing the fault. Similarly, before the service is re-established, the *BackupActivate* message needs to traverse the length of the end-to-end backup in the former scheme and the length of a backup segment in the latter (see Fig. 7). Thus, failures are handled more locally and quickly in our scheme. To state it quantitatively, the failure-recovery delay is proportional to the lengths of primary and backup paths. If there are k segments in the backup, it results in $O(k)$ improvement. This could be a substantial improvement for real-time applications over many hops, which cannot tolerate long durations of service disruption. Though our routing algorithm *Min_SegBak* does not share minimizing recovery delay as one of its design goals, we believe that it is possible to design heuristics that provide better failure-recovery time guarantees at the cost of greater resource utilization.

B. Applications in the Current Internet

In today's Internet, our primary-segmented backup scheme can be implemented both at the network level in the routers and at the application level among a set of end hosts forming an application level overlay.

[1]We are making a qualitative observation here based on the fact that most end-to-end path lengths in the Internet are less than 30.

1) At the network level: It can be run within a single autonomous system (AS) by Internet backbone providers like UUNET [25] or Internet service providers to guarantee QoS to their customers. It is well known that Internet backbone providers attempt to design their physical networks to ensure that there are disjoint paths between any two routers. Also, by confining the scheme to within a single AS, one can ignore any sort of scalability concerns.
2) At the application level: It can be run at the application level among a set of routers forming a logical network (overlay) over existing physical inter-network such as in resilient overlay networks (RON) [1], detour [23], or reliable backbone (RBONE) [4]. Recent studies on Internet routing stability by Labovitz *et al.* [13] have shown that in the current inter-domain routing protocol, Border Gateway Protocol (BGP) [22], failure-recovery mechanisms take on the order of tens of minutes to stabilize the routing tables. This has resulted in a growing research interest in providing QoS guarantees at the application layer over an overlay network. For example, in RON, a set of end hosts in the Internet run a custom routing protocol (which can provide QoS guarantees) between themselves. Our scheme can be applied in such scenarios for achieving more efficient QoS guarantees. Employing our scheme at the overlay level frees us from the deployability concerns with resource reservation schemes such as RSVP in the current Internet.

VI. Performance Evaluation

We evaluated the performance of our scheme by carrying out simulation studies on regular network topologies such as meshes of size 5×5, 7×7, 9×9, and 12×12, as well as real-world network topologies like the USANET. These experiments are similar to those used in [6] and [8] to evaluate the performance of end-to-end backups. The network simulator was written in C++ and run on a PC with a Pentium-II 400-MHz processor. We also implemented the end-to-end backup scheme as described in [6] and [8] to compare its performance with that of our scheme in terms of the amount of spare resources reserved and the average call acceptance rate (ACAR) (this represents the fraction of requested calls that are accepted when averaged over a long duration of time) at various network loads.

In all our network topologies, neighbor nodes are connected by two simplex links, one in each direction, and all links have identical bandwidth and delays. The bandwidth of each link is chosen as 40 units, while the delay is set to one unit. Thus, the delay along any path is proportional to its length. The experiments consist of establishing a large number of dependable connections between pairs of nodes. For every connection, we route the primary and backup channels as follows.

1) Primary channels are routed from source to destination using Dijkstra's shortest-path algorithm over links having sufficient bandwidth as required by the connection. In the case of multiple shortest paths, the primary is routed over any one of the paths without any explicit preference given to paths closest to the boundary of network topology, as was done in [6]. The routing in [6] was designed specifically to exploit the mesh network topology to find two disjoint paths and cannot be used for topologies such as the USANET.
2) We route the end-to-end and segmented backup paths as follows.
   a) For the end-to-end backup, all components of the primary path (i.e., all the links and the intermediate nodes) are removed and a shortest path search algorithm is used to route over links having sufficient bandwidth as required by the connection.
   b) For the segmented backup, the *Min_SegBak* algorithm is used to route the backup segments over links having sufficient bandwidth as required by the connection.

When both the primary and the backup are routed successfully, resources are reserved along them using deterministic backup multiplexing under single-link failure model (as described in the spare resource aggregation algorithm given in Section III).

When any component of a primary segment fails, its corresponding backup segment is activated and the primary path is minimally rerouted only around the failed primary segment. In Fig. 1, we show the path after recovering from failure of link 4. If the faulty component belongs to two successive primary segments, any one of the two backup segments corresponding to those segments is activated. Note that *every end-to-end backup is a special case of a segmented backup with one backup segment*. Below, we use simple illustrations over a 5×6 mesh topology to capture the intuitive advantages of segmented backups that motivated our work. We chose the mesh topology primarily for simplicity in presentation and the scenarios discussed here can be shown to occur over more realistic topologies such as the USANET (see Fig. 10). The experiments are run for a large number of time units. One connection is requested every time unit between a pair of nodes chosen randomly from the set of all possible pairs of nodes. The bandwidth requirement of all requested connections is set to one unit. Further, every established connection is torn down after a fixed number of time units called *Call Duration*. The network is allowed to reach a stable state before any results are noted. When the network is at a stable state, the number of active connections is proportional to *Call Duration*. Thus, by varying the bandwidth of the links and the *Call Duration*, we can subject the network to varying levels of load. The graphs shown in this section are plotted for loads (measured as percentage of total network bandwidth resource on all the links reserved for both primary and backups taken together) varying from 5% to 70%.

We use a metric called *average hop count* in all our plots to compare the amounts of resources reserved for various paths. The average hop count of primary paths is computed as *the sum of the lengths of all the primary paths in the network divided by the total number of active connections*. Average hop counts for end-to-end backups and segmented backups are calculated similarly with the exception that whenever two backups requiring one unit of bandwidth each are multiplexed on a link, only one unit is added to the sum total. It is important to note that in

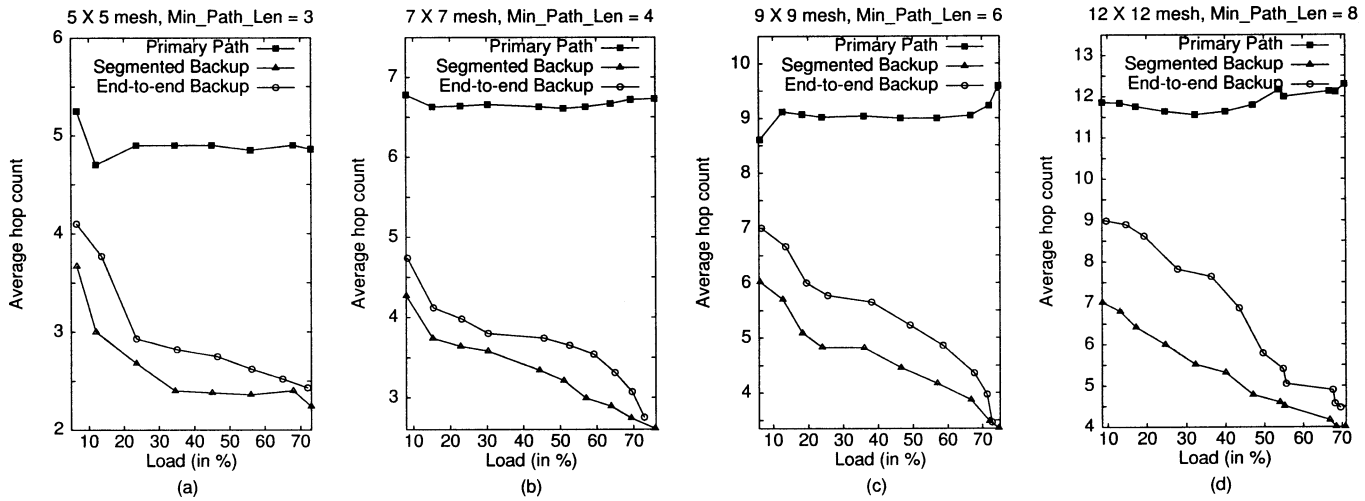

Fig. 8. Comparing the amount of spare resources reserved by segmented backup and end-to-end backup schemes over mesh topologies of various sizes. (a) 5×5 mesh. (b) 7×7 mesh. (c) 9×9 mesh. (d) 12×12 mesh.

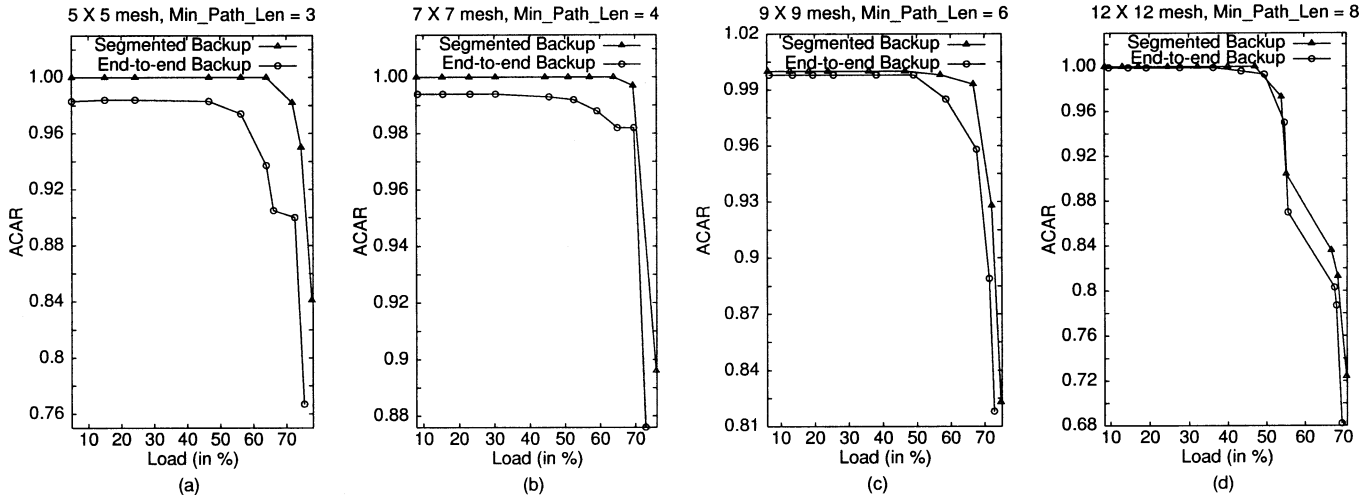

Fig. 9. Comparing the ACAR for segmented backup and end-to-end backup schemes over mesh topologies of various sizes. (a) 5×5 mesh. (b) 7×7 mesh. (c) 9×9 mesh. (d) 12×12 mesh.

our experiments, the average hop counts of primary and backup paths are directly proportional to the amount of resources reserved for the paths as every connection has the same one unit of bandwidth requirement. For example, suppose that at 40% network load, the average hop counts of primary and segmented backup are 12 and 6, respectively. This implies that while 40% of the total network resources are reserved, two-thirds $(12/(12+6))$ of the reserved resources are allocated for primary paths (i.e., 27% of the total network resources) while one-third $(6/(12+6))$ of the reserved resources are allocated for backup paths (i.e., 13% of the total network resources).

We expect the advantages of using our scheme over end-to-end backup schemes to increase as the length of the primary increases. This is because longer primary paths have more backup segments and all of the advantages that go with them. To capture this effect in our study, we introduced a parameter called *Min_Path_Len* and requested only connections between nodes that have the length of the shortest path between them greater than *Min_Path_Len*. We chose *Min_Path_Len* depending on the size and diameter of the network topology. For square meshes of sizes 5, 7, 9, and 12, we chose *Min_Path_Len* to be 3, 4, 6, and 8, respectively, while for USANET we chose it to be 0. These values are reasonable as a significant percentage (USANET—100%, 5×5 mesh—66%, 12×12–52%) of node pairs in these network topologies have the length of the shortest path between them greater than the chosen value of *Min_Path_Len*.

In Fig. 8, we compare the resource reservation requirements of our scheme with that of the end-to-end backup scheme for meshes of varying sizes. Similarly, we compare the ACAR of the two schemes in Fig. 9. Finally, in Fig. 11, we compare the performance of the schemes over USANET. We now go into a detailed analysis of the results in each of these figures.

Comparing the Amount of Spare Resources Reserved: The graphs in Fig. 8(a)–(d) show the average hop counts of primary path, end-to-end backup and segmented backup at various network loads for meshes of sizes 5×5, 7×7, 9×9, and 12×12, respectively. Note that for the average hop count of primary paths, we plot a single curve rather than separate curves for the two schemes. In our simulations, we found that the hop counts

TABLE I
AVERAGE NUMBER OF BACKUP SEGMENTS FOR VARYING MESH SIZES

Mesh Size	5 X 5	7 X 7	9 X 9	12 X 12
Segments	1.26	1.39	1.49	1.59

of primary paths for the schemes differ by as little as 4%. This prompted us to plot only a single curve as it facilitates comparison of the relative performances of the schemes. The following characteristics can be easily spotted.

1) The spare resources required by either scheme are considerably less than the resources required for the primary path.
2) As the network load increases, the average hop count for primary path varies very little, while it decreases steadily for backups.
3) Our scheme always requires a lesser amount of spare resources than the end-to-end scheme. This difference in resources reserved is quite significant at low and intermediate loads (30%–45%) but decreases toward high loads.
4) As we go to larger networks, from 5×5 to 12×12:
   a) The average hop count of the primary path increases considerably and the number of backup segments per segmented backup increases as shown in Table I
   b) The hop count difference between end-to-end backup and segmented backup increases.

We now attempt to explain the observed characteristics. Observation 1 is a direct consequence of backup multiplexing, which allows backups to share reserved resources. As the network load increases, more backup paths are active simultaneously, which improves the chances for multiplexing. This increased multiplexing explains the decrease in the average hop count (proportional to resources reserved) for backups as noted in observation 2. Also, increasing the network load alters the number of primary paths that are active simultaneously but not their length, so their average hop count hardly varies. As we argued in Section III, backup segments are shorter than end-to-end backup segments, which improves the chance of multiplexing in the former over the latter. This explains observation 3, where we notice increased savings in resources reserved for our scheme. However, it is not possible to share more and more resources by the way of multiplexing indefinitely and these additional savings decrease at high loads, when the network reaches saturation (The graphs showing ACAR in Fig. 9 indicate that the networks saturate at 70%–75% load).

Observation 4(a) is expected as larger networks allow longer connections to be established. With increasing primary path lengths, it is natural to expect an increase in the number of backup segments. This is shown in Table I, where there is a steady increase in the average number of backup segments for increasing mesh sizes at moderate network loads (40%–50%).[2] We can explain our observation 4(b) as follows. As the average number of backup segments increases, the backup segments tend to be increasingly shorter compared to their end-to-end counterparts. This improves the chance of multiplexing in segmented backups over end-to-end backups, resulting in greater difference in hop counts and more savings in spare resources reserved. This difference in resources reserved improves from 15% in 5×5 mesh to 25% in 12×12 mesh.

Comparing the ACAR: The graphs in Fig. 9(a), (b), (c), and (d) show the ACAR curves for end-to-end backup and segmented backup schemes at various network loads for meshes of sizes 5×5, 7×7, 9×9, and 12×12, respectively. The following characteristics can be observed.

1) The ACAR curves are stable and high for small network loads and then drop suddenly and steeply.
2) The difference in the curves for 12×12 mesh is negligible, while the difference is noticeable (ours gives better ACAR) for smaller networks especially 5×5 mesh; this trend is exactly opposite to what was observed in spare resource reservation.
3) While our scheme gives ACAR $= 1.000$ until the network load increases to 50%-55%, the end-to-end scheme never gives ACAR $= 1.000$ even when the network load is as low as 5%.
4) Our scheme reaches saturation at slightly higher network loads.

The high ACAR values noted in observation 1 for both the schemes until the network is heavily loaded are expected, as any mesh topology has a large number of alternate routes between any two nodes. Almost all calls are accepted until the network reaches saturation. This also accounts for the negligible difference between the ACAR curves for segmented and end-to-end backups in a 12×12 mesh. Both schemes score very high ACAR until they reach saturation at about 70% load. However, the ACAR improvement in smaller networks comes because of the scenario illustrated in Fig. 2(a) and generalized in Theorem 1. There exist node pairs such that end-to-end backups do not exist for a chosen primary path between them but segmented backups do exist. The probability of encountering such a scenario decreases rapidly with the increase in the size of the network. This explains our observations 2 and 3. Finally, a probable explanation for observation 4 is that our scheme accepts more calls by reserving less resources, thereby saturating at a higher load than the end-to-end scheme.

Comparing the Performance of Segmented Backup and End-to-End Backup Schemes Over USANET: In Fig. 11(a) and (b), we plot the relative performance of segmented backup and end-to-end backup schemes over the USANET topology shown in Fig. 10, using average hop count and ACAR as metrics. We note the following.

1) The average hop count of the primary path of connections established is very low (3–3.5).
2) The curves for spare resource reservation between the two schemes are almost inseparable.
3) There is significant and uniform improvement in the call acceptance rate. Even at very low loads, the ACAR for end-to-end backup scheme never goes above 0.945.

Observation 1 is explained by the fact that the network is small, with just 28 nodes, and that *Min_Path_Len* was set to 0. With the small average primary path length (< 4), a vast majority of the calls will have backups with only one segment (same as end-to-end backup). Thus, the resource resevation by

[2]The average number of backup segments is less than two in all meshes due to our choices of network topology and load for the simulations.

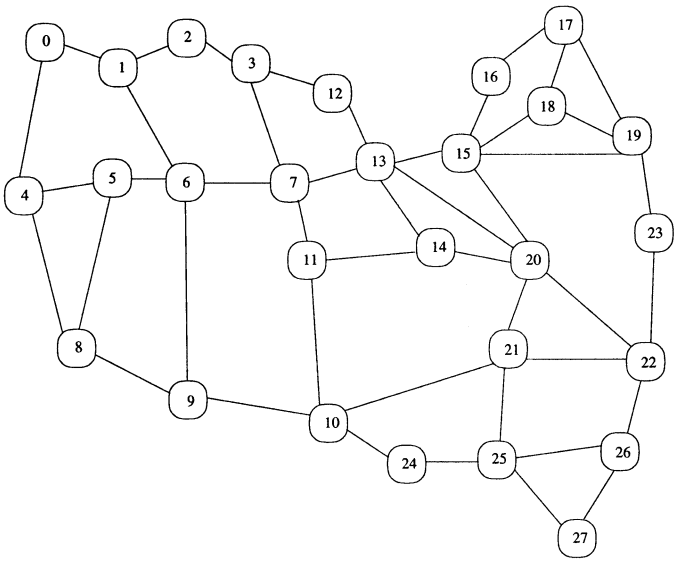

Fig. 10. The 28-node topology of the USANET.

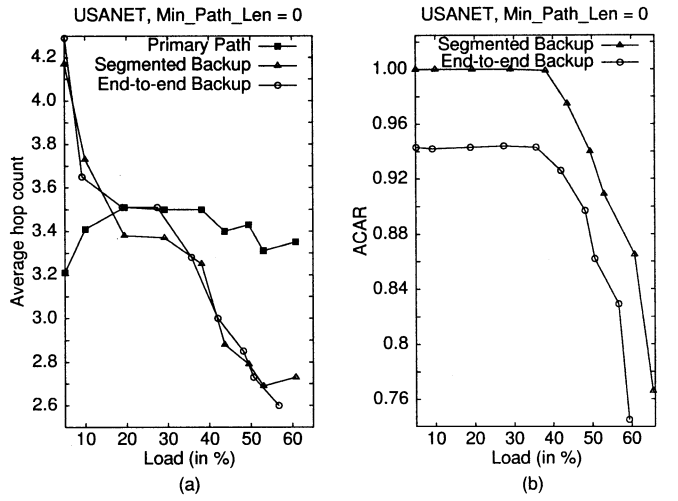

Fig. 11. Comparing the performance of segmented backup and end-to-end backup schemes over USANET topology.

both schemes is similar as pointed in observation 2. Finally, as observation 3 points out, about 6% of the requested calls are rejected because of lack of an end-to-end backup in scenarios where a segmented backup can be found.

To summarize, our simulations demonstrate that our scheme is capable of delivering better resource efficiency as well as better call acceptance rate compared with existing end-to-end backup schemes. However, the benefits of our scheme vary with the network topology and load. Our scheme performs significantly better for larger networks with low connectivity (more nodes and less edges) at low and moderate loads.

VII. Conclusion

In this paper, we have proposed segmented backups: a failure-recovery scheme for dependable real-time communication in multihop networks. This mechanism not only improves resource utilization and call acceptance rate but also can provide faster failure recovery and better QoS guarantees on end-to-end delays without compromising the level of fault tolerance provided. We have also given an efficient backup route selection algorithm. We evaluated the proposed scheme through extensive simulations and demonstrated the superior performance of our method compared to earlier schemes.

In order to realize the full potential of the method of segmented backups, routing strategies must be developed for backup segments to achieve better QoS guarantees on delay and bounded time failure recovery.

Appendix I
Proof of Theorem 1

Theorem 1

Whenever two disjoint paths exist between a source and a destination in a network, segmented backups are guaranteed to exist for any choice of primary path between the end nodes. However, there are no such guarantees for end-to-end backups.

Proof: In Fig. 2(a), we demonstrated a network topology where disjoint paths exist between a pair of end nodes but end-to-end backups do not exist for a chosen primary path. Here, we prove that segmented backups exist whenever disjoint paths exist. We start with a simple observation. We refer to the backup segment spanning a primary segment that contains an intermediate node N as a backup segment covering N. For example, in Fig. 1, the three backup segments over links *A–C*, *D–G*, and *H–K* cover the nodes $N1$–$N2$, $N3$–$N5$, and $N6$–$N8$, respectively. Observe that a segmented backup for a primary path P can be constructed by taking the set of all backup segments covering each of the intermediate nodes in P, as in Fig. 1. Below, we prove our claim of existence of a segmented backup by showing the existence of backup segments that cover every intermediate node.

In our graph $G(V, E)$, let the two disjoint paths between source S and destination D be denoted by P_1 and P_2, respectively. Let P be any chosen primary path between them and let $len(P)$ denote its length. Two cases arise:

- Case 1) $len(P) = 1$ (i.e., P has only one edge $e(S, D)$). This is the special case with no intermediate nodes. One of P_1 or P_2 is a segmented backup for $\mathbf{P}$, as edge E cannot be in both P_1 and P_2.
- Case 2) $len(P) > 1$ (i.e., P has at least 1 intermediate node). Let N denote any intermediate node on P. We need to show the existence of a backup segment that covers N. Since P_1 and P_2 are disjoint, at least one of them does not contain N. Without loss of generality let us assume that N does not lie on P_1. We claim that since 1) P and P_1 have the same end points S and D, and 2) $N \in P$ and $N \notin P_1$, a segment (a contiguous subpath) of P_1 acts as a backup segment covering N. We prove it using recursion.

Base Case for Recursion: P and P_1 are disjoint. Clearly, P_1 is a suitable backup segment covering the primary segment P containing N.

Recursive Step: We apply it when P and P_1 are not disjoint. We show the existence of sub paths P' and P_1' for paths P and P_1 (i.e., $len(P') < len(P)$ and $len(P_1') < len(P_1)$) such that 1) P' and P_1' have the same end points and 2) $N \in P'$ and $N \notin P_1'$.

Let $P = S, i_1, i_2, \ldots, i_r, \ldots, i_k(= N), \ldots, D$ and $P_1 = S, j_1, j_2, \ldots, j_s, \ldots, D$, denote the nodes along the paths with i_r and j_s representing the rth and sth vertices along the paths P and P_1 respectively. N is the kth intermediate node on the path P. As P and P_1 are not disjoint, they must have some common node N' such that $N' = i_{r_1} = j_{s_1}$ for some r_1, s_1. As node $N \notin P_1$ either $r_1 < k$ or $r_1 > k$. In either case, we define P' and P'_1 as follows: If $r_1 < k$, $P' = i_{r_1}, i_{r_1+1}, \ldots, i_k = N, \ldots, D$ and $P'_1 = j_{s_1}(= i_{r_1}), j_{s_1+1}, \ldots, D$. If $r_1 > k$ then, $P' = S, i_1, i_2, \ldots, i_k = N, \ldots, i_{r_1}$ and $P'_1 = S, j_1, j_2, \ldots, j_{s_1}(= i_{r_1})$. Clearly, 1) paths P' and P'_1 have same end points and 2) $N \in P'$ and $N \notin P'_1$. Further, $len(P') < len(P)$ and $len(P'_1) < len(P_1)$.

If P' and P'_1 are disjoint the base case assures us of the existence of a backup segment covering N. If not, this step is applied recursively. Since the paths are of finite lengths and decrease in each iteration, this process always terminates in the base case. Thus, every node along the primary path is guaranteed a backup segment that covers it and a segmented backup can be generated by taking all the backup segments together. ∎

APPENDIX II
PROOF OF THEOREM 2

First, we state and prove two lemmas. We use them later to prove the theorem.

Lemma 1

The weight (cost) of the segmented backup, i.e., the sum of weights of all backup segments generated, is equal to the weight of the shortest path *B* found in step 2 of the algorithm.

Proof: Every directed edge in B which does not point to a vertex in $P - \{D\}$ is included in one of the backup segments by steps 3b and 3c. We replaced every edge which starts from a vertex not in P but points to a vertex in $P - \{D\}$ with an edge of equal weight in step 3d(i). This leaves us with the case of edges between vertices along primary path P. Edges from a vertex in P to its successor vertex are included in backup segments by step 3d(ii), while edges from a vertex in P to its predecessor vertices are excluded. However, these excluded edges are of zero weight and do not contribute to the weight of the path. Finally, there are no extra edges added to the backup segments. This is illustrated in Fig. 5(b) and (c), where both the shortest path B and segmented backup weigh six units. Thus, we conclude that the weight of the segmented backup generated is equal to the weight of path *B*. ∎

Lemma 2

Every segmented backup for primary path P between S and D in G can be mapped to a path between the same nodes in G' that is of equal weight.

Proof: For any chosen segmented backup for P in G, we show the existence of a path in G' between the same end nodes that is of equal weight. Suppose the chosen backup consists of b backup segments $BS_1, \ldots, BS_i, \ldots, BS_b$. Denote the corresponding primary segments as $PS_1, \ldots, PS_i, \ldots, PS_b$. Let $PS_{i,f}$, $PS_{i,l}$ and $PS_{i,l-1}$ denote the first, last, and penultimate nodes of the ith primary segment, respectively. Similarly, let $BS_{i,f}$, $BS_{i,l}$ and $BS_{i,l-1}$ denote the first, last, and the penultimate nodes of the ith backup segment, respectively. Clearly, $PS_{i,f} = BS_{i,f}$, $PS_{i,l} = BS_{i,l} \forall i$. Also, note that the first and the penultimate nodes of a segment with only a single link are the same.

To show the existence of a path from $PS_{1,f}(= S)$ to $PS_{b,l}(= D)$ in G' that is of equal weight, we claim that: 1) there exists a path from $PS_{i,f}$ to $PS_{i,l-1}$ that is of same weight as $BS_i \forall i < b$; 2) there exists a path from $PS_{b,f}$ to $PS_{b,l}$ that is of same weight as BS_b; and 3) there exists a path from $PS_{i,l-1}$ to $PS_{i+1,f}$ that is of zero weight $\forall i < b$.

All edges of the backup segments that do not point to any vertex in $P - \{D\}$ are left unchanged while modifying G into G' by step 1 of the algorithm. Thus only the last edge in BS_i is changed $\forall i < b$ and no edge is changed in BS_b. For each BS_i this last edge from $BS_{i,l-1}$ to $BS_{i,l}(= PS_{i,l})$ is redirected to point to $PS_{i,l-1}$. Thus, in the modified graph G' the edges in BS_i form a path from $PS_{i,f}$ to $PS_{i,l-1} \forall i < b$ and the edges in BS_b form a path from $PS_{b,f}$ to $PS_{b,l}$. This proves our claims 1 and 2 above. As successive primary segments overlap over at least one edge of the primary path, either $PS_{i+1,f} = PS_{i,l-1}$ or $\mathrm{pred}(PS_{i+1,f}, PS_{i,l-1})$. From step 1a of the algorithm, we know that there is zero weight path from any node on primary path P to its predecessors. This proves our claim 3 above.

By taking the edges of the paths in claims a, b and c above, we obtain a single path from $PS_{1,f}(= S)$ to $PS_{b,l}(= D)$ in G' that is of same weight as the segmented backup.

We now use the Lemmas 1 and 2 to prove Theorem 2. ∎

Theorem 2

The segmented backup generated by the *Min_SegBak* algorithm is a minimum cost segmented backup for any chosen primary path.

Proof: To prove the theorem we need to show that: 1) the backup generated is a valid segmented backup; and 2) it is a minimum-cost segmented backup. While we avoid formal arguments for 1) here, one can easily prove it by establishing the converse of Lemma 2, namely, every path between S and D in G' can be mapped to a valid segmented backup in G by following step 3 of the algorithm. Instead, we assume 1) holds and prove 2). It follows from Lemma 2 that the weight of the shortest path in G' between S and D is at most the weight of the minimum-cost segmented backup. However, Lemma 1 states that the weight of the segmented backup generated by our algorithm equals the weight of the shortest path between S and D in G'. Thus, we conclude that the weight of the segmented backup generated by *Min_SegBak* is at most the weight of minimum-cost segmented backup. ∎

REFERENCES

[1] D. G. Anderson *et al.*, "Resilient overlay networks," in *Proc. 18th ACM Symp. Operating System Principles*, Banff, Canada, Oct. 2001, pp. 131–145.

[2] J. Anderson, B. Doshi, S. Dravida, and P. Harshavardhana, "Fast restoration of ATM networks," *IEEE J. Select. Areas Commun.*, vol. 12, pp. 128–139, Jan. 1994.

[3] A. Banerjea, "Simulation study of the capacity effects of dispersity routing for fault-tolerant real-time channels," in *Proc. ACM SIGCOMM*, Aug. 1996, pp. 194–205.

[4] C. Dovrolis and P. Ramanathan, "Resource aggregation for fault-tolerance in integrated services networks," in *Proc. ACM SIGCOMM Comput. Commun. Rev.*, Apr. 1998, pp. 39–53.
[5] W. Grover, "The self-healing network: A fast distributed restoration technique for networks using digital crossconnect machines," in *Proc. IEEE GLOBECOM*, 1987, pp. 1090–1095.
[6] S. Han and K. G. Shin, "Efficient spare-resource allocation for fast restoration of real-time channels from network component failures," in *Proc. IEEE Real-Time Systems Symp.*, 1997, pp. 99–108.
[7] ——, "Experimental evaluation of failure detection schemes in real-time communication networks," in *IEEE Fault-Tolerant Computing Symp. Dig. Papers*, 1997, pp. 122–131.
[8] ——, "A primary-backup channel approach to dependable real-time communication in multihop networks," *IEEE Trans. Comput.*, vol. 47, pp. 46–61, Jan. 1998.
[9] K. Ishida, Y. Kakuda, T. Kikuno, and K. Amano, "A distributed routing protocol for finding two node-disjoint paths in computer networks," *IEICE Trans. Commun.*, vol. E82-B, no. 6, pp. 851–858, June 1999.
[10] B. Kao, H. Garcia-Molina, and D. Barbara, "Aggressive transmissions of short messages over redundant paths," *IEEE Trans. Parallel Distrib. Syst.*, vol. 5, pp. 102–109, Jan. 1994.
[11] R. Kawamura, K. Sato, and I. Tokizawa, "Self-healing ATM networks based on virtual path concept," *IEEE J. Select. Areas Commun.*, vol. 12, pp. 120–127, Jan. 1994.
[12] G. P. Krishna, M. J. Pradeep, and C. S. R. Murthy, "A segmented backup scheme for dependable real-time communication in multihop networks," in *Proc. 8th IEEE Workshop Parallel and Distributed Real-Time Systems*, May 2000, pp. 678–684.
[13] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian, "Delayed internet routing convergence," in *Proc. ACM SIGCOMM*, Aug. 2000, pp. 175–187.
[14] G. Manimaran, H. S. Rahul, and C. S. R. Murthy, "A new distributed route selection approach for channel establishment in real-time networks," *IEEE/ACM Trans. Networking*, vol. 7, pp. 698–709, Oct. 1999.
[15] A. Mankin *et al.*, "Resource reservation protocol (RSVP)," IETF, RFC 2208, 1997.
[16] J. Moy, "OSPF protocol analysis," IETF, RFC 1245, 1991.
[17] ——, "OSPF Version 2," IETF, RFC 2328, 1998.
[18] C. Parris and D. Ferrari, "A dynamic connection management scheme for guaranteed performance services in packet-switching integrated services networks," Univ. California, Berkeley, Tech. Rep. TR-93-005, 1993.
[19] M. J. Pradeep and C. S. R. Murthy, "Providing differentiated reliable connections for real time communicaion in multihop networks," in *Proc. Int. Conf. High Performance Computing (HiPC)*, Dec. 2000, pp. 459–468.
[20] P. Ramanathan and K. G. Shin, "Delivery of time-critical messages using a multiple copy approach," *ACM Trans. Comput. Syst.*, vol. 10, no. 2, pp. 144–166, May 1992.
[21] G. Ranjith, G. P. Krishna, and C. S. R. Murthy, "A distributed primary-segmented backup scheme for dependable real-time communication in multihop networks," in *Proc. IEEE Int. Parallel and Distributed Processing Symp.*, Apr. 2002, pp. 139–146.
[22] Y. Rekhter and T. Li, "A Border Gateway Protocol 4 (BGP-4)," IETF, RFC 1771, 1995.
[23] S. Savage *et al.*, "Detour: A case for informed internet routing and transport," *IEEE Micro*, vol. 19, pp. 50–59, Jan. 1999.
[24] R. Sriram, G. Manimaran, and C. S. R. Murthy, "A rearrangeable algorithm for the construction of delay-constrained dynamic multicast trees," *IEEE/ACM Trans. Networking*, vol. 7, pp. 514–529, Aug. 1999.
[25] (2001) Network Maps. UUNET Technologies. [Online]. Available: http://www.uunet.com/network/maps
[26] R. Vogel *et al.*, "QoS-based routing of multimedia streams in computer networks," *IEEE J. Select. Areas Commun.*, vol. 14, pp. 1235–1244, Sept. 1996.
[27] Z. Whang and J. Crowcroft, "Quality-of-service routing for supporting multimedia applications," *IEEE J. Select. Areas Commun.*, vol. 14, pp. 1228–1234, Sept. 1996.
[28] Q. Zheng and K. G. Shin, "Fault-tolerant real-time communication in distributed computing systems," in *IEEE Fault-Tolerant Computing Symp. Dig. Papers*, 1992, pp. 86–93.

Krishna Phani Gummadi received the B.Tech. degree in computer science and engineering from the Indian Institute of Technology, Madras, India, in 2000. He is currently working toward the Ph.D. degree in computer science and engineering at the University of Washington, Seattle.

His research interests include Internet measurement studies, design and analysis of scalable systems, distributed systems, and real-time systems.

Mr. Gummadi is a student member of the Association for Computing Machinery. He is a corecipient of the Best Paper Award from the Multimedia Computing and Networking Conference, San Jose, CA, in 2002.

Madhavarapu Jnana Pradeep received the B.Tech. degree in computer science and engineering from the Indian Institute of Technology, Madras, India, in 2000 and the M.S. degree in computer science and engineering from the University of Illinois, Urbana-Champaign, in 2002.

He is currently a Software Design Engineer with the Distributed Storage and File Systems Group, Microsoft Corporation, Redmond, WA. His research interests include operating systems, real-time systems, ubiquitous computing, and software engineering.

C. Siva Ram Murthy (M'97–SM'02) received the B.Tech. degree in electronics and communications engineering from Regional Engineering College, Warangal, India, in 1982, the M.Tech. degree in computer engineering from the Indian Institute of Technology (IIT), Kharagpur, India, in 1984, and the Ph.D. degree in computer science from the Indian Institute of Science, Bangalore, India, in 1988.

He joined the Department of Computer Science and Engineering, IIT, Madras, as a Lecturer in September 1988, and became an Assistant Professor in August 1989 and an Associate Professor in May 1995. He has been a Professor with the same department since September 2000. He has held visiting positions at the German National Research Center for Information Technology (GMD), Sankt Augustin, Germany, the University of Washington, Seattle, the University of Stuttgart, Germany, and the EPFL, Switzerland. He has published over 150 research papers in international journals and conferences. He is a coauthor of the textbooks *Parallel Computers: Architecture and Programming* (New Delhi, India: Prentice-Hall of India, 2000), *New Parallel Algorithms for Direct Solution of Linear Equations* (New York: Wiley, 2001), *Resource Management in Real-Time Systems and Networks* (Cambridge, MA: MIT Press, 2001), and *WDM Optical Networks: Concepts, Design and Algorithms* (Upper Saddle RIver, NJ: Prentice-Hall, 2002). His research interests include parallel and distributed computing, real-time systems, lightwave networks, and wireless networks.

Dr. Murthy is a recipient of the Best Ph.D. Thesis Award and the Indian National Science Academy Medal for Young Scientists. He is a corecipient of Best Paper Awards from the 5th IEEE International Workshop on Parallel and Distributed Real-Time Systems, Geneva, Switzerland, in 1997 and the 6th International Conference on High Performance Computing, Calcutta, India, in 1999. He is a Fellow of the Indian National Academy of Engineering.