A Distributed Primary-Segmented Backup Scheme for Dependable Real-Time Communication in Multihop Networks

G. Ranjith
Department of Computer Science and Engg.
Indian Institute of Technology, Madras
Chennai, India 600036
g_ranjith@yahoo.com

G. P. Krishna
Department of Computer Science and Engg.
University of Washington
Seattle, WA 98195-3530
gphanikrishna@hotmail.com

C. Siva Ram Murthy
Department of Computer Science and Engg.
Indian Institute of Technology, Madras
Chennai, India 600036
murthy@iitm.ac.in

Abstract

Several distributed real-time applications require fault-tolerance apart from guaranteed timeliness, at acceptable levels of overhead. These applications require hard guarantees on recovery delays, due to network component failures, which cannot be ensured in traditional datagram services. Several schemes exist which attempt to guarantee failure recovery in a timely and resource efficient manner. These methods center around a priori reservation of network resources called spare resources along a backup channel, in addition to each primary communication channel. This backup channel is usually routed along a path disjoint with the primary channel. In this paper, we propose a distributed method of segmented backups for dependable real-time communication in multihop networks, which improves upon existing methods in terms of network resource utilization, average call acceptance rate, scalability and provides better QoS guarantees on bounded failure recovery time and propagation delays, without any compromise in fault-tolerance levels. The distributed algorithm is one of finding a "minimal path" based on flooding with a cut-out mechanism that does not relay messages if they came along longer paths than those known. We further show that the complexity of the distributed algorithm is bounded and acceptable.

1 Introduction

The rapid development of high speed networking technology has made possible a plethora of 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. The applications outlined above fall into the class of distributed real- time applications, where the deadlines associated with a data element is as important as the data element itself. Hence, a minimum "quality of service" (QoS) such as delay of communication, availability and error-rate, is to be contracted with the user(s), for these applications. Any communication network is prone to faults due to hardware failure or software bugs. Hence, it is essential to incorporate fault-tolerance capabilities into these QoS contracts, especially as these applications last for long durations, thus increasing the probability of a fault occurring in its life-time. Unlike conventional schemes, real-time applications require special schemes to meet the hard QoS guarantees like bounded message delays. In these schemes, resources (link bandwidth, buffer space) are reserved a priori along some path, and all messages of the real-time session follow this path. We shall call this path through the network, between a source node (which sends data) and the destination node (which receives data), as the primary path. But this brings on the issue of failure of components along this predetermined path.

1.1 Related Work

A simple and effective method to ensure fast and guaranteed failure recovery is to reserve a priori spare resources along another path, called a backup *path (or channel)[1, 2, 3], which is disjoint with the primary path. Activation of these spare resources occur only when the pri-*

Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)
1530-2075/02 $17.00 © 2002 **IEEE**

mary channel is disabled. End-to-end detouring *[2] is a simple solution to this in which an end-to-end backup channel (i.e., a backup channel extending from source to destination) is set up along with each primary channel. But this scheme has several drawbacks, the main ones of which are:* **1:** *The backup path should be* totally disjoint *with the primary path. As the primary path is decided first and then another disjoint path is selected as the backup path, the primary path can be routed (so that it minimizes some metric like delay or hop-length) in such a way that it "blocks out" another end-to-end backup, as shown in Figure 2 (explained later).* **2:** *This method has the further restriction that* both *the primary and the backup paths separately satisfy all the QoS requirements like end-to-end delay etc. This is a very hard restriction, even when there are enough free network resources.*

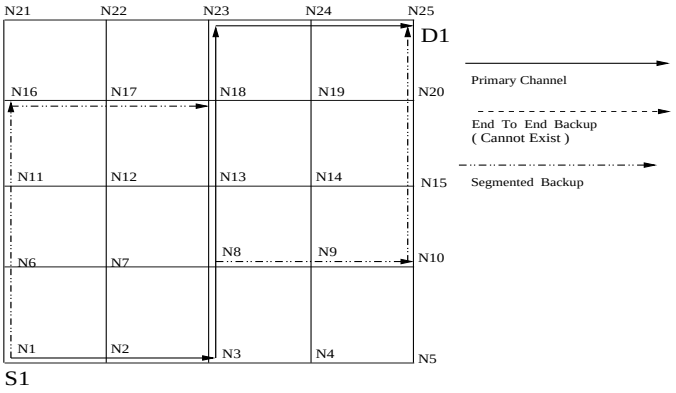

Figure 2. No end-to-end backup but segmented backup exists

1.2 The Segmented Backups Approach

A novel and realizable scheme that solves most of the problems that are briefly outlined above is the segmented backups *scheme, first proposed in [4]. In [5], we elaborate this scheme, and present a better backup route selection algorithm. Extensive simulations had also been done in that paper to bring out the effectiveness of the scheme. We shall now summarize the work reported in [5]. The basic idea of segmented backups is to provide backups to the primary path as segments, rather than a continuous disjoint path from source to destination. It has been shown that this method not only solves the problems mentioned above, but also is very resource efficient. In this paper, we realize the segmented backups scheme in a fully distributed and scalable fashion. Note that in this paper, the first two sections are a brief overview of [5].*

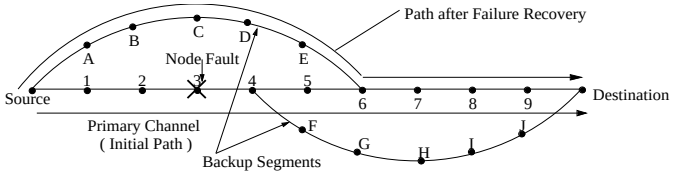

Figure 1. Illustration of segmented backups

Earlier schemes have used end- to-end backups, i.e., *backups which run from source to destination of a dependable (fault-tolerant real-time) connection, with the restriction that the primary and the backup channels are disjoint. In our approach of* segmented backups, *we find backups for only parts of the primary path. The primary path is viewed as made up of smaller contiguous paths, which we call* primary segments *as shown in Figure 1. 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. Consider Figure 1, where a network of nodes and links are shown, and where a dependable connection is established between the nodes marked* source *and* destination. *The primary channel runs through the links connecting the nodes* source, *1 to 9 and* destination. *There are two backup segments involved in this setup. The first one runs through the links that connect the nodes* {source,A,B,C,D,E,6} *while the second one runs through the links connecting the nodes* {4,F,G,H,I,J,destination}. *The first backup segment "backups" the primary segment given by the nodes (and the links connecting them)* {source,1,2,3,4,5,6} *while the second backup segment "backups" the primary segment that runs through the nodes* {4,5,6,7,8,9,destination}. *Note that we have the condition that the primary segments should always overlap by at least one link.*

Now let us briefly consider that there is a failure at any one of the nodes on the primary path. Note that any such node is always on one (or more) primary segment *and there will be one (or more) backup segment that is the "backup" for this/these primary segment(s). For example, in Figure 1, when node 3 fails, the first backup segment is* activated, i.e., *the traffic flows along the new path shown in Figure 1. If there are more than one backup segment for that particular node (as in node 5), we can choose the segment that gives the best QoS (e.g., delay, jitter, error-bound) guarantees.*

We shall illustrate the main advantage of backups segments with Figure 2. Here, a dependable connection is requested between the nodes N1 and N25, in a 4 × 4 mesh. The primary path happens to be established as shown. Now, we note that with this primary path, there will be no primary-disjoint backup paths possible whatsoever. *But a segmented backup path is possible, as shown in the figure. We have shown in [5] that this happens in real-life topologies like the 28-node USANET.*

Now we define the exact criteria for selecting the set of

Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)
1530-2075/02 $17.00 © 2002 **IEEE**

links that constitute the segments of the segmented backups. Note that we are choosing two things: 1) the number of backup segments used and 2) the links that constitute each such backup segment. Let us define the weight of a single segment as sum of the hop counts (or any suitable metric) of the links that constitute that backup segment (note: this excludes any link from the primary path). Now, we define the weight of a segmented backup system as below:

$$\textit{Weight-Segmented-Backup} = \sum_{i=1}^{n} Weight - of - Segment$$

where n is the number of segments. Now, we choose the segmented backup system which gives a minimum value of this weight*. In other words, we try to minimize the Weight-Segmented-Backup parameter. And henceforth, we shall call the set of segmented backups that minimize this weight as the* minimum segmented backup*. What we achieve by using this criteria is a segmented backups system that consumes minimum extra resources, per primary channel. Note that we have mainly concentrated on the resource-reservation issues of real-time communication and do not address issues like message scheduling or queuing at each node. Here, it can be safely assumed that the real-time model (which specifies the real-time issues like queuing and scheduling) assumed in [3] will be applicable in our case, too. This is because in that work too, the authors try to achieve QoS guarantees based on end-to-end resource reservations.*

We note that end-to-end backup is also a case of segmented backup with the number of segments equal to one. Segmented backups tend to be more resource-efficient than end-to-end backups due to the fact that they are less constrained than the former. Hence, there is more freedom to choose backup paths (that may involve multiple segments) that are more resource-efficient than end-to-end backup paths. Also, we explain briefly why using segmented backups is better than using end-to-end backups, as far as deterministic backup multiplexing is concerned. Note that two backup segments can be multiplexed when their primary segments *are disjoint. Primary segments tend to be smaller than the whole primary path. Hence, there is a greater probability that they will be disjoint, as smaller paths are more probable to be disjoint.*

In a later section, we shall see how we shall achieve distributed backup multiplexing by keeping a minimal level of information at each node, for each link associated with that node.

The rest of this paper is organized as follows. In Section 2, we briefly outline how the backup route selection is done. In Section 3, we give the distributed version of the algorithm used in Section 2. Advantages gained in failure recovery is mentioned in Section 4. We briefly discuss about the implementation of our scheme in current computer networks in Section 5. We conclude our work in Section 6.

2 Backup Route Selection

In this section, we shall describe exactly how we select the set of links that will serve as the segments of the segmented backup system, so that we get a minimum segmented backup system (see Section 1.2).

Algorithm *Min_SegBak:*

Let the directed graph $G(V, E)$ represent the given network topology. Every node n in the network is represented by a unique vertex v in the vertex set V and every duplex link l between nodes $n_1(v_1)$ and $n_2(v_2)$ in the network is represented in the graph G by two directed edges e_1 and e_2 from v_1 to v_2 and v_2 to v_1, respectively.

Let S and D denote the source and destination nodes in the network. We denote a primary path (already found out) in graph G with a sequence of vertices $P = S, i_1, i_2, ..., i_n, D$. In order to find the shortest segmented backup, we generate a modified graph G' as follows:

1. *Every directed edge $e(v_i, v_j)$ other than those along the primary path (i.e., edges between any two successive vertices in the sequence P) is assigned the weight w_{ij}.*
2. *For edges along the primary path, the weights are assigned as follows: Edges directed from a vertex in the sequence P to its successor vertex are assigned a weight of infinity (edge is removed). Edges directed from a vertex in the sequence P to its predecessor vertex are assigned a weight of zero (see Figure 3).*
3. *For every edge $e(v_1, v_2) \in E$, if $v_1 \notin P$ and $v_2 \in (P - S)$, replace e with $e'(v_1, v_2')$ where v_2' is the predecessor of v_2 in P. That is, replace every edge from any vertex v_1 not in P, directed into any intermediate vertex v_2 in P, with another edge directed from v_1 to v_2', predecessor of v_2.*

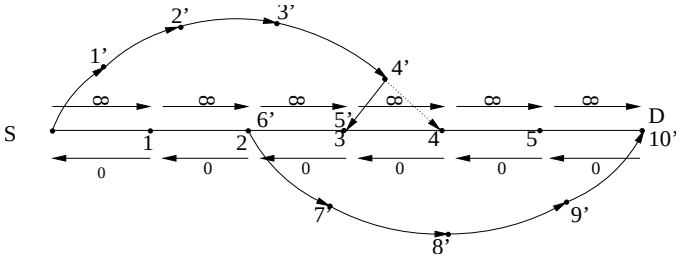

Figure 3. Illustration of the construction of the shortest segmented backup from the path chosen

To find the shortest segmented backup, on the resulting graph G',

Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)
1530-2075/02 $17.00 © 2002 **IEEE**

4. *Run the least weight path algorithm for directed graphs (e.g., Dijkstra's algorithm) from the source to destination on G'. Let the path obtained be denoted by a sequence of vertices $B = S, i'_1, i'_2$ ...., i'_m, D.*

5. *Assume that the segmented backup consists of backup segments BS_1, BS_2.... As we traverse the sequence B from S to D, we generate the vertex sequences for these segments BS_1, BS_2, BS_3, .. one after the other. The phrase open a segment indicates the beginning of generation of that backup segment, and close a segment to indicate the ending of the generation of that backup segment. Hence, in our algorithm we first open BS_1, generate it, close it, then open BS_2, generate it, close it and so on, until all the backups segments are generated. At any stage of the traversal, if there is an opened backup segment being generated, then it is denoted as current backup segment. If all the opened segments are closed, current backup segment is $NULL$. The vertex sequences $BS_1, BS_2, ...$ are initialized to be empty when opened. The phrase add vertex v to a sequence means the vertex is appended at the end of the sequence.*

   For constructing backup segments, we traverse the sequence B (found in step 4). At every stage of the traversal, let i'_c denote the current vertex. We perform the appropriate actions as indicated in (a) to (d) below, for every i'_c. This procedure ends on reaching D.

   (a) If $i'_c = S$ then open BS_1 and add i'_c to it.

   (b) If $i'_c \neq i_k$ for any $k \leq n$, (i.e., i'_c does not lie on P) then

      i. If current backup segment $\neq NULL$ then add i'_c to current backup segment.

      ii. If current backup segment = $NULL$ then open next backup segment and add i'_{c-1} and i'_c to it in that order.

   (c) If $i'_c = i_k$ for any $k \leq n$, (i.e., i'_c lies on P) then

      i. If current backup segment $\neq NULL$ then add i_{k+1} to current backup segment and close it.

      ii. If current backup segment = $NULL$ do nothing

   (d) If $i'_c = D$ then add i'_c to current backup segment and close it.

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

We explain step 5 of our algorithm in Figure 3. Here, the primary path runs through nodes 1 through 5. Suppose the path chosen between S and D in G' is over the nodes numbered $1'$ through $10'$. We denote by a dotted line, the edge between $4'$ and 4 in G which is replaced in step 3 with an edge between $4'$ and 3(=$5'$). The backup segments are generated as follows: First, we open BS_1 and add S as given in case (a). Then we add $1'$ through $4'$ in succession to BS_1, as given in sub case(i) of case(b). Then when we traverse $5'$(=3) we add 4 and close BS_1 as given in sub case(i) of case(c). Then we ignore $6'$ as given in sub case(ii) of case(c). Then when we come to $7'$, we open BS_2 and add $6'$ and $7'$ to it as given in sub case(ii) of case(b). Then we add $8'$, $9'$ and $10'$ as before, before closing BS_2 with D as given in case(d).

A proof that the Min_SegBak algorithm given above finds the minimum segmented backups possible is found in [5]. The intuition is that as we have found the minimum path in G', it will correspond to the minimum segmented backups.

3 The Distributed *Min_SegBak* Algorithm

In this section, we show how the complex routing function of the Min_SegBak algorithm can be distributed efficiently. The fundamentals of distributed network protocols, along with an acceptable general model (that this paper uses) and several examples can be found in [6].

In a distributed environment, nodes in the network model used are only aware of the local network topology, i.e., the identity of its neighbors. Minimal message passing is done between these neighbors, and no broadcasting is used. The distributed algorithm has to be executed twice, i.e., the first time for finding a minimal cost primary path between the source and the destination, and the second time for finding a set of minimal total weight segmented backups for this primary path. After the two iterations of the algorithm has been completed, the nodes on the primary and the backup segments know that they are so, and also know their predecessors *and* successors *(explained later). The algorithm (each iteration) runs in two distinct phases: Phase I implements running of a shortest path algorithm on the graph G' described in the previous sections. Phase II is initiated by the source and it updates the state information at all the nodes in which it has to be done so.*

As explained above, Phase I implements the running of a shortest path algorithm on the graph G'. We do this in the following manner: 1) Take a well-known and proven distributed shortest-path algorithm that finds the shortest path (from a node "source" to all other nodes) on a given graph G, and 2) Modify it in such a way that it finds this shortest path in G', and not in G. This path would correspond to some path in G. On termination of our algorithm, the state variables would reflect these paths. Now for point 1 above, we choose the distributed shortest path algorithm presented in [7] as it is simple and effective. Coming to point 2, we

Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)
1530-2075/02 $17.00 © 2002 **IEEE**

note that it can be affected by changing the way in which messages are sent and received in the original algorithm. At the end of the running of our modified shortest path algorithm, all nodes are aware of the neighbor to use as a predecessor to get to the source in the shortest distance, with respect to the modified graph G'.

3.1 Description of State Variables

Consider a network of nodes corresponding to a directed graph G, with a computational process p_i running at each node v_i, and v_i and v_j are neighbors if edge (v_i, v_j) exists in the network. The term process, vertex and node are used interchangeably here. We classify every neighbor of i in the following way: For each node i of G, we use the notation out_nbrs_i to denote the "outgoing neighbors" of i, i.e., those nodes to which there are edges from i in the digraph of G, and in_nbrs_i to denote the "incoming neighbors" of i. Now we describe how the information about the primary path and the backup segments to this primary path, of a particular dependable connection, are stored in the system. We shall use Figure 1 for further clarifications.

Each node i stores the following three variables: $member\_communication\_paths$, $pred\_list$ and $succ\_list$. The variable $member\_communication\_path$ is a boolean that is $true$ if node i is a member of either the primary path or any of the backup segments. The variable $succ\_list$ is an ordered set of nodes that can have a cardinality of either zero, one or two. Hence, $succ\_list$ can be either $\{NULL\}, \{sl_1\}, \{sl_1, sl_2\}$. For a node that is either on a primary path or on a backup segment (or both, as is the case of node 4 of Figure 1), this set represents the set of nodes that succeeds *it. Node 4 is succeeded by node 5 on the primary path and node F on the second backup segment. Hence, $succ\_list$ will be set to $\{5,F\}$ at node 4. Note that node 5 is before node F in this list, and as a rule, nodes on the primary path are put at the beginning. Hence, this is an ordered set, with the first element representing the node along the primary path. Note that for a node that lies only on the primary path or only on a backup segment, the cardinality is equal to one. One such example is node 7 (in Figure 1), where $succ\_list$ is equal to $\{8\}$. If a node is neither on the primary path nor on any backup segment, $succ\_list$ is equal to $NULL$.*

The variable $pred\_list$ is similar to the $succ\_list$ variable. But, in this case, it represents the predecessor to this node on either the primary path or on any backup segment, and has similar properties.

We also note that for this given graph G, there exists a unique graph G' that is constructed as described before. The $distance$ of a vertex v_i is the length of the shortest path from S to v_i with respect to G' and is denoted by L_i. p_i knows the weight w_{ij} for every outgoing (v_i, v_j) and incoming (v_j, v_i) edges. We note that without loss of generality, we take the source node as $v1$ and denote the process running at it as $p1$. Process p_1 running at the source initiates the computation to determine the $distance$ metric L_i at all vertices. Refer to [7] for a similar approach. Also note that the main "node algorithm"[6] is subdivided into nine modules ($M1$ to $M9$) depending on the input and the type of the node.

3.2 The Structure of Phase I Computation

Messages used in phase I:

1. A length message is a two-tuple (s, p), where p is the identity of the process (or node) sending the message and s is a real number. p_i sends a message (s, p_i) to p_j to inform that there is a path of length s from v_1 (source) to v_j in which v_i is the prefinal vertex.

2. An acknowledgment message or ack, which has no other data associated with it. A process p_j sends an ack to a process p_i in response to a length message sent by p_i, which means that the $length$ message sent by p_i has been, or will be taken care by all processes reachable from p_j.

A process p_i maintains a local variable d which denotes the length of the shortest path received so far by p_i. Upon receiving a length s from a predecessor, if $s < d$, p_i sets d to s and in this case it sends a $length$ message $(d + w_{ij}, p_i)$ to every successor p_j.

Local data used by a process p_i during phase I:

Each process p_i uses three local variables, for phase I:

d: This is the shortest length of paths from v_1 (source) to v_i known to this process at this point of time with respect to the modified graph G'. $d = \infty$ if no $length$ message has been received.

pred_to_S_in_G' : This is the predecessor from which the $length$ message corresponding to d was received; it also is the prefinal vertex on the shortest path to v_i computed so far. $pred\_to\_S\_in\_G'$ is undefined if $d = \infty$ or i=1.

num: This is the number of unacknowledged messages, that is, the number of messages sent by this process for which no ack has been received so far.

Phase I algorithm for process p_i, $i \neq 1$

M1: *Initialization:*
01: {*No $length$ messages have been received; there*
02: *are no unacknowledged messages*}
03: **begin** $d := \infty$*; $pred\_to\_S\_in\_G'$ is undefined;*
04: $num := 0$ **end**;

M2: *Upon receiving a $length$ message (s, p_j):*
01: **If** $s < d$ **then**
02: **begin**
03: **if** $member\_shortest\_paths = true$ **then**

Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)
1530-2075/02 $17.00 © 2002 **IEEE**

04: {*We have to define successor in such a way*
05: *so that it reflects the topology of G'*}
06: **if** *(i = D)* {*node is Destination*} **then**
07: $successor := NULL$
08: **else if** $j \neq sl_1 of succ\_list$ **then**
09: $successor := pred\_list$
10: **else** $successor := out\_nbrs_i$
11: $- succ\_list + pred\_list$
12: *For node* $k \in pred\_list$, *use* $w_{ik} = 0$
13: *(only for the execution of this module* **M2***)*
14: **else**
15: $successor := out\_nbrs_i$
16: {*Send an ack to* $pred\_to\_S\_in\_G'$, *the prefinal*
17: *vertex on the previous* $shortest$ *path, if it has*
18: *not been already sent*}
19: **if** $num > 0$ **then** *send an* ack *to* $pred\_to\_S\_in\_G'$
20: {*update* $d, pred\_to\_S\_in\_G'$}
21: $pred\_to\_S\_in\_G' := p_j; d := s;$
22: {*Send a* $length$ *message to all successors of* v_i
23: *and increment* num *appropriately and then*
24: *return* ack *to* $pred\_to\_S\_in\_G'$ *if* num*=0*}
25: *Send* $length(d + w_{ik}, p_i)$ *to every* $successor$ p_k.
26: num*:=* num *+ the number of successors of* v_i.
27: **If** num*=0* **then** *send an* ack *to* $pred\_to\_S\_in\_G'$
28: **end**
29: **else** $\{s \geq d\}$ {*new* $length$ *message does not denote*
30: *a shorter path*}
31: *Send* ack *to* p_j.

M3: *Upon receiving an* ack *from process* p_k
01: **begin**
02: {*decrement number of unacknowledged*
03: *messages*}
04: num*:=* num *- 1;*
05: {*send acknowledgment to* $pred\_to\_S\_in\_G'$ *if*
06: $acks$ *have been received for all messages*}
07: **if** num*=0* **then** *send an* ack *to* $pred\_to\_S\_in\_G'$
08: **end**
Note: If $num > 0$ *at any time, this basically means that a process has* $exactly$ *one message to which it has not sent an* ack, *and this* ack *should go to* $pred\_to\_S\_in\_G'$.

Phase I algorithm for process p_1, the source
M4: *Initialization*
01: d*:=0;* $pred\_to\_S\_in\_G'$ *is undefined;*
02: {*We have to define successor in such a way so*
03: *that it reflects the topology of G'*}
04: $successor := out\_nbrs_1 - succ\_list$
05: *send* (w_{1k}, p_1) *to all successors* p_k; num*:=*
06: *number of successors of* v_1.

M5: *Upon receipt of a length message* (s, p_i) *from any other node:*
01: *return* ack *to* p_i

M6: *Upon receiving an* ack
01: {*Update* num*; start phase II if there is no*
02: *unacknowledged messages remaining*}
03: num *:=* num*-1;*
04: **if** num*=0* **then** *terminate phase I and send*
05: $start\_phase$ *message to destination.*

3.3 The Structure of Phase II Computation

Messages used in phase II:
1. A $start\_phase$ *message has no other data associated with it. It is sent by the source to the destination on termination of phase I.*
2. A $trace\_back$ *message, which has no other data associated with it. This is sent originally by the destination to the node that precedes it on the newly-found path. It traces its way back to the source and appropriately modifies the information at the nodes to reflect the knowledge about the backup segments.*

Phase II algorithm for process $Destination$
M7: *Upon receiving a start_phase from source*
01: **begin**
02: {*Send* $trace\_back$ *message to the predecessor;*
03: *This starts the trace-back process* }
04: *Send* $trace\_back$ *to* $pred\_to\_S\_in\_G'$
05: **end**

Phase II algorithm for process p_i, $i \neq 1$
M8: *Upon receiving a update_state from process* p_j
01: **begin**
02: {*Node type is of node 5 in Figure 1: There is*
03: *no change of any state variable at this node* }
04: **If** $j \in pred\_list$ *and* $pred\_to\_S\_in\_G' \in succ\_list$
05: **then**
06: $NOperation$
07: {*Node is of node 6 in Figure 1: Add node*
08: $pred\_to\_S\_in\_G'$ *into* $pred\_list$ }
09: **If** $j \in pred\_list$ *and* $pred\_to\_S\_in\_G' \notin succ\_list$
10: **then**
11: $pred\_list := pred\_list - j + pred\_to\_S\_in\_G'$
12: {*Node type is of node 4 in Figure 1: Add* j
13: *to* $succ\_list$ }
14: **If** $j \notin pred\_list$ *and* $pred\_to\_S\_in\_G' \in succ\_list$
15: **then**
16: $succ\_list := succ\_list + j$
17: {*Node type is of node* H *or* D *in Figure 1: Add*
18: *this node as a new node to the primary-backup*
19: *system, now it is part of the primary path or a*
20: *backup segment*}

Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)
1530-2075/02 $17.00 © 2002 **IEEE**

21: **If** $j \notin pred\_list$ *and* $pred\_to\_S\_in\_G' \notin succ\_list$
22: **then**
23: $member\_shortest\_paths := true$
24: $pred\_list := pred\_list + pred\_to\_S\_in\_G'$
25: $succ\_list := succ\_list + j$
26: {*In any case, pass on the message to the*
27: *next node* }
28: *Send* $trace\_back$ *to* $pred\_to\_S\_in\_G'$
29: **end**

Phase II algorithm for process p_1, the source
M9: *Upon receiving a trace_back from process* p_j
01: **begin**
02: {*Add that node to the* $succ\_list$*; Now the*
03: *whole algorithm is completed* }
04: $succ\_list := succ\_list + j$
05: **end**

3.4 Complexity Analysis

For the distributed Min_SegBak algorithm, we assume that any message takes at most time d for traversal of the transmission channel between the two nodes involved (as it is the case with most modern-day communication systems). In that case, the time bound for the algorithm can be calculated as $O(nd)$. A more rigorous analysis of the algorithm can be obtained by arguing on the same lines as one would argue for the Asynchronous Bellman-Ford Algorithm [8]. We also note that in the worst case, the total communication complexity will be exponential in n. This is the same case for the time complexity. But the worst case is quite unlikely in this case, and the the actual number of messages and the actual time in the average case will be much smaller.

Note that only a few routers on the network have to store extra information (those at the beginning or end of the segments), and hence the number of states is not greatly increased.

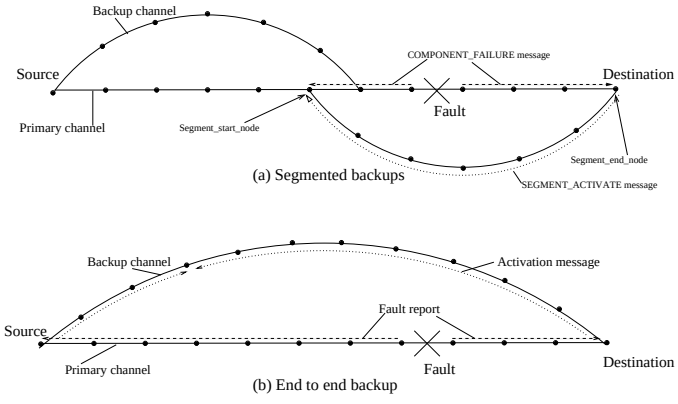

Figure 4. Illustration of failure recovery

4 Distributed Failure Recovery

When a fault occurs in a component in the network, all dependable connections passing through it have to be rerouted through their backup paths. This process is called failure recovery. This has the following phases: fault detection, failure reporting, *and* backup activation.

The time taken for reestablishing service is equal to the sum of the times taken by each of the above phases, and is called failure recovery delay. This delay is crucial to many real-time applications and has to be minimized.

Delay of failure recovery is the turn-around time that is required for migrating the traffic from the failed component to the backup segment. We note that this is lower in our scheme, because the recovery can be local and messages have to traverse only the length of the segment involved, rather than the whole distance from the source to the destination, as would have been the case with end-to-end backups. This can be seen from Figure 4. The total end-to-end delay along the path of a real-time connection is another important QoS parameter for real-time communication which has to be minimized. Note that there will be an increase in this delay when we switch from the primary path to a backup path. This increase in the end-to-end delay that a dependable connection will encounter can be termed as the failure delay increment. We note that this parameter will also be lower in our scheme, due to the more localized failure recovery. We note that localized failure recovery is an added advantage of our scheme.

When a fault occurs, packets transmitted during the failure reporting time are lost. In our scheme, when a fault occurs in one segment of the primary, only the packets which have entered that segment from the time of the occurrence of the fault till the backup segment activation are lost. This is in contrast to the end-to-end backup case, where all *packets in transit are lost.*

5 Implementation of our Scheme

In this section, we first describe the relevant scenarios in the current Internet and other places where our scheme can be used and then discuss the protocols required to implement it.

Applications of Our Scheme in Current Internet: *Our primary-segmented backup scheme can be used in two different scenarios in today's Internet.*

1. *It can be run at the network level between the backbone routers by Internet backbone providers like UUNET [10] or between the routers in a single autonomous system (AS) by any Internet service provider. 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.*

Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)
1530-2075/02 $17.00 © 2002 **IEEE**

2. *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) [11], Detour [12].*

Applications of Our Scheme in Other Multihop Networks: *It is also important to note that the scheme can be implemented in general on any multihop networks, in addition to the Internet. This can also include VPN(virtual private networks) based networks that are logically disassociated from the Internet, even if they depend on the Internet at the lower network layers. We also note that as computer communications become more mission critical, QoS-demanding and involve more real-time data, there is a growing interest in "private networks", i.e., multihop networks that are dedicated to a select set of users and which can provide the required levels of QoS and availability. These networks should be self-contained and should be free to deploy any routing and signaling protocols, regardless of the Internet.*

Resource Reservation Protocol: *The Resource Reservation Protocol (RSVP) [9] was the first solid proposal for resource-reservation based schemes for the Internet and for general multihop networks. It has been outlined in detail in [5] how RSVP can be used for the resource reservation and reconfiguration phases of our scheme.*

6 Conclusions

In this paper, we presented an efficient, scalable and distributed method for providing single-component fault-tolerant communication between nodes in a communication network. The concept of segmented backups, as opposed to that of of end-to-end backups for providing this fault-tolerance has been presented and its superiority has been discussed. Then, an algorithm by which we can select the exact links that will participate as redundant resources, was presented. Later in the paper, it was shown how this algorithm could be implemented in a distributed fashion. We have discussed issues like failure recovery and complexity of the proposed algorithm. It has been noted how this algorithm is relevant both to the current Internet and also for other multihop networks in today's scenario where both QoS and network availability are becoming major issues. It has also been briefly discussed how our algorithm can be easily implemented on top of current resource reservation protocols like RSVP.

References

[1] Q. Zheng and K. G. Shin, "Fault-Tolerant Real-Time Communication in Distributed Computing Systems," in Proc. of IEEE FTCS, *pp. 86-93, 1992.*

[2] R. Kawamura, K. Sato, and I. Tokizawa, "Self-healing ATM Networks Based on Virtual Path Concept," IEEE Journal on Selected Areas in Communications, *Vol. 12, No. 1, pp. 120-127, January 1994.*

[3] S. Han and K. G. Shin, "A Primary-Backup Channel Approach to Dependable Real-Time Communication in Multihop Networks," IEEE Trans. on Computers, *Vol. 47, No. 1, pp. 46-61, January 1998.*

[4] G. P. Krishna, M. J. Pradeep and C. Siva Ram Murthy, "A Segmented Backup Scheme for Dependable Real-Time Communication in Multi-Hop Networks", in Proc. of IEEE WPDRTS, pp. 678-684, 2000.

[5] G. P. Krishna, M. J. Pradeep, and C. Siva Ram Murthy, "An Efficient Primary-Segmented Backup Scheme for Dependable Real-Time Communication in Multihop Networks", Revised Version Communicated to IEEE/ACM Trans. on Networking, 2001.

[6] A. Segall, "Distributed Network Protocols", IEEE Trans. on Information Theory, Vol. 29, No. 1, pp. 23-35, January 1983.

[7] K. M. Chandy and J. Misra, "Distributed Computation on Graphs: Shortest Path Algorithms", Communications of the ACM, Vol. 25, No. 11, pp. 833-837, November 1982.

[8] N. A. Lynch, Distributed Algorithms, *Morgan Kaufmann Publishers, 1997.*

[9] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A New Resource Reservation Protocol", IEEE Network, Vol. 7, No. 5, pp. 8-18, September 1993.

[10] UUNET, "UUNET Technologies", http://www.uunet.com/network/maps, *October 2001.*

[11] D. G. Anderson, H. Bala Krishnan, M. F. Kashooek, and R. Morris, "Resilient Overlay Networks," in Proc. of ACM SOSP, *October 2001.*

[12] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and John Zahorjan, "Detour: A Case for Informed Internet Routing and Transport," IEEE Micro, *Vol. 19, No. 1, pp. 50-59, January 1999.*

[13] G. Manimaran, H. S. Rahul, and C. Siva Ram Murthy, "A New Distributed Route Selection Approach for Channel Establishment in Real-Time Networks", IEEE/ACM Trans. on Networking, *Vol. 7, No. 5, pp. 698-709, 1999.*

Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)
1530-2075/02 $17.00 © 2002 **IEEE**