# A New Distributed Route Selection Approach for Channel Establishment in Real-Time Networks

G. Manimaran, *Member, IEEE,* Hariharan Shankar Rahul, and C. Siva Ram Murthy, *Member, IEEE*

***Abstract*—In this paper, we propose a new distributed route selection approach, called *parallel probing*, for real-time channel establishment in a point-to-point network. The existing distributed routing algorithms fall into two major categories: preferred neighbor based or flooding based. The preferred-neighbor approach offers a better call acceptance rate, whereas the flooding approach is better in terms of call setup time and routing distance. The proposed approach attempts to combine the benefits of both preferred neighbor and flooding approaches in a way to improve all the three performance metrics simultaneously. This is achieved by probing $k$ different paths in parallel, for a channel, by employing different heuristics on each path. Also, the proposed approach uses a notion called *intermediate destinations* (ID's), which are subset of nodes along the *least-cost* path between source and destination of a call, in order to reduce the excessive resource reservations while probing for a channel by releasing unused resources between ID's and initiating parallel probes at every ID. Further, it has the flexibility of adapting to different load conditions by its nature of using different heuristics in parallel, and hence, a path found for a channel would have different segments (a segment is a path between two successive ID's), and each of these segments would very well be selected by different heuristics. The effectiveness of the proposed approach has been studied through simulation for well-known network topologies for a wide range of quality-of-service and traffic parameters. The simulation results reveal that the average call acceptance rate offered by the proposed route-selection approach is better than that of both the flooding and preferred neighbor approaches, and the average call setup time and routing distance offered by it are very close to that of the flooding approach.**

***Index Terms*— Channel establishment, distributed routing, heuristics, quality of service, real-time networks.**

## I. Introduction

PACKET-switched data networks are increasingly being utilized for carrying multimedia traffic such as video and audio which often require stringent quality-of-service (QoS) requirements in terms of end-to-end delay, delay jitter, and loss. For a network to provide performance guarantees for such multimedia applications, *real-time channels* [7] are to be established with specified traffic characteristics and QoS requirements. The traffic characteristics include parameters such as maximum message rate, maximum message size, and maximum burst size. There are two phases involved in handling a real-time channel: channel establishment [9] and run-time scheduling [9], [19] of packets. The channel establishment phase involves the selection of a *qualified* route for the channel satisfying traffic characteristics and QoS requirements of the call request, without compromising the guarantees of the existing channels. Although several channel establishment schemes have been proposed, very few of them have explicitly addressed the issue of route selection, despite its importance in the channel establishment phase.

There are two basic approaches to the route selection problem: centralized and distributed. In a centralized route-selection approach, the existence of a global network manager is assumed, which maintains information about all the established real-time channels and the network topology, and can thus select an appropriate route for each real-time channel request. In such a centralized approach, every real-time channel request has to be approved by the network manager. Although this is better in terms of selecting a qualified route by employing an efficient algorithm for network management, it suffers both in terms of performance and reliability due to the use of a centralized network manager. In contrast with the centralized approach, the distributed route selection approach offers better performance and is more reliable. Since the number of routes between a source and destination is very high, choosing a qualified route is not an easy task. The objective of any routing algorithm is to find a qualified path with minimal operational overheads. The distributed routing problem, in the context of real-time channel establishment, has been studied by some researchers [3], [8], [10], [15], [17] in the recent past. The primary objective of these algorithms has been to improve the call acceptance rate without considering the response time taken to set up a connection and the routing distance (hops) taken by the connection. In this paper, we propose a new distributed routing approach whose objective is not only improving the call acceptance rate, but also minimizing the call setup time and the routing distance.

There are two broad schemes for real-time channel establishment: single-pass and two-pass schemes [14]. In a two-pass scheme [7], there are two phases, namely, the reservation phase (forward pass) and relaxation phase (reverse pass). The forward pass proceeds from source to destination of the channel request (call) during which, at each intermediate node, a call admission test (which depends on the scheduling

Manuscript received October 23, 1997; revised March 29, 1999 and June 21, 1999; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor A. U. Shankar.

G. Manimaran was with the Department of Computer Science and Engineering, Indian Institute of Technology, Madras, India. He is with the Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50011 USA (e-mail: gmani@iastate.edu).

H. S. Rahul was with the Department of Computer Science and Engineering, Indian Institute of Technology, Madras, India. He is now with the Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139 USA (rahul@lcs.mit.edu).

C. Siva Ram Murthy is with the Department of Computer Science and Engineering, Indian Institute of Technology, Madras 600 036, India (e-mail: murthy@iitm.ernet.in).

Publisher Item Identifier S 1063-6692(99)08527-1.

algorithm used) is performed to check whether the resources such as bandwidth, buffers, and delay guarantee required by the call can be satisfied; these resources are reserved if the call admission test is successful. Once the forward pass is successful, the destination initiates the reverse pass. During the reverse pass, the resources which were allocated excessively during the forward pass are relaxed so that these excess resources can be allocated to some other calls. If the call is rejected, the resources that were reserved along the path found so far are released. Since the two-pass scheme does not reserve more resources than that are required by a call, the average call acceptance rate (ACAR) offered by a routing algorithm using the two-pass scheme is higher than its single-pass counterpart. On the other hand, the single-pass scheme offers low average call setup time (ACST). We believe that the call acceptance rate is more important than call setup time. Therefore, in this paper, we consider the two-pass scheme for real-time channel establishment. In a two-pass scheme, it is necessary to reserve resources during forward pass so that other calls cannot reserve the same resources during their forward pass. This avoids the possibility of two or more calls attempting to confirm the same resource during their reverse pass.

The rest of the paper is organized as follows. In Section II, we discuss the existing routing approaches, their weaknesses, and motivations for our work. The proposed distributed routing approach is presented in Section III. In Section IV, the performance of the proposed routing approach is compared with that of the existing routing algorithms through simulation studies. Finally, in Section V, some concluding remarks are made.

## II. Background and Motivation

The network, in which the routing problem is addressed here, is an arbitrary point-to-point topology. In a real-time network, at any point in time, many channel establishment requests are active, whose objective is to find a qualified routing path from their respective sources to destinations. Path selection within routing is typically formulated as a shortest path optimization problem. The objective function may be number of hops, cost, delay, or some other metric that corresponds to a numeric sum of the individual link parameters along a selected path [10]. Efficient algorithms (Dijkstra and Bellman–Ford) exist for computing shortest paths in communication networks. However, within the context of satisfying diverse QoS requirements, the algorithms become more complex as constraints are introduced in the optimization problem. These constraints typically fall into two categories: link constraints and path constraints [3], [10]. A link constraint is a restriction on the use of links on a path based on the available capacity (such as bandwidth) of the links which must be greater than or equal to that required by the call, whereas path constraint is a bound on the combined value of a QoS parameter along a selected path (such as end-to-end delay offered along the path must not exceed what the call can tolerate). Path constraints make a routing problem intractable. A shortest path problem, even with a single path constraint, is known to be NP-complete [6]. Therefore, heuristics are usually employed to solve the QoS routing problem.

### A. Performance Metrics

In a traditional computer network, most of the routing algorithms tend to optimize the message delay or the route distance for a single connection [8], [15]. To optimize the global performance of a real-time network, the following metrics are to be optimized.

For an accepted call request "$R$" let us define the functions:

- accepted$(R) = 1$
- setup$(R)$ = number of nodes visited by the call setup packet
- dist$(R)$ = length of the path (in terms of hop count) chosen for $R$

For a call request $R$ that is rejected, all the functions return a value of zero. Let "$N$" be the total number of call-requests generated.

- **Average Call Acceptance Rate ( ACAR):** the probability of accepting a real-time channel establishment (call) request, defined as the ratio of number of calls accepted to the number of calls arrived in the system

$$\text{ACAR} = \frac{\sum_{i=1}^{N} \text{accepted}(R)}{N}.$$

- **Average Call Setup Time (ACST):** the average time required to setup a real-time channel measured in terms of number of nodes visited by the call setup packet

$$\text{ACST} = \frac{\sum_{i=1}^{N} \text{setup}(R)}{\sum_{i=1}^{N} \text{accepted}(R)}.$$

- **Average Routing Distance (ARD):** the average hop count of the established channels

$$\text{ARD} = \frac{\sum_{i=1}^{N} \text{dist}(R)}{\sum_{i=1}^{N} \text{accepted}(R)}.$$

The first metric is very important as it is a measure of call throughput. The second metric is crucial in the context of real-time and interactive multimedia applications, which require fast channel setup. The third metric is essential because a shorter route is less costly. In the context of real-time communication, traditional metrics, such as average message delay and message throughput, used in datagram networks are meaningless since they do not necessarily indicate anything about the timeliness of messages.

### B. Earlier Work on Route Selection

Several heuristic routing algorithms have been proposed for the real-time channel establishment problem [8], [10], [15], [17]. The existing distributed routing algorithms fall into two major categories: flooding based and preferred-neighbor

based. In a flooding-based approach, a packet is forwarded to all (or some) of its neighbors, except the node from which the packet has come, to find a qualified route. Whereas in a preferred neighbor approach, a packet is forwarded to a preferred neighbor which is chosen based on a heuristic, such as shortest path first (SPF) or lightly loaded link first (LLF).

The flooding-based approach is superior in terms of ACST and ARD at the cost of ACAR. The lower ACST and ARD of the flooding approach is due to its nonbacktracking nature, since all the paths are probed simultaneously, whereas its poor call acceptance is due to excessive reservation of resources, such as bandwidth and buffers, along many paths from source to destination during the forward pass of a call setup. In comparison, the preferred-neighbor approach offers higher ACAR at the cost of ACST and ARD. Its higher ACAR is due to reservation of resources along only one path as opposed to multiple paths in the flooding-based approach, and its higher (poor) ACST and ARD are due to its backtracking nature (backtracking happens when there is no qualified path from the current node to the destination through all or some of the preferred neighbors of the current node).

A distributed route-selection scheme based on the flooding approach has been proposed in [15] for route selection during real-time channel establishment. In this approach, the number of messages used for establishing a call is at most $2K$, where $K$ is the number of links in the network. This is very expensive and results in tentatively reserving resources in many nodes and links, thereby reducing the ACAR. Sending multiple copies of a message to the destination in order to meet the deadline (end-to-end delay) of the message has been proposed in [11]. The intent of this approach is to send a message as quickly as possible, and it is not meant for real-time channel establishment. If this approach is extended to real-time channel establishment, it will be a variation of flooding, which is not efficient due to its poor ACAR. A distributed routing algorithm, called selective probing, was proposed in [4], wherein probes (called establishment packets) are flooded selectively along those paths which satisfy the QoS and optimization requirements. An improvement to selective probing was proposed in [5]. These algorithms do not aim at maximizing all the three performance metrics.

There are two types of preferred neighbor heuristics: 1) local/static knowledge based and 2) dynamic nonlocal knowledge based. Algorithms such as SPF and LLF are examples for the first type. The overhead associated with these algorithms is almost the same since these heuristics either use local link information or relatively static global information. For example, the preferred neighbor table of LLF is based on local link information, whereas that of SPF is based on relatively static topology of the network.

Heuristics, such as least-delay first [13], [16], is an example of the second type. Here, the preferred neighbor table for a destination is updated based on the delays offered by other nodes along the paths, from the current node to the given destination. Since the delay is dynamically varying, it has to be propagated from one node to another, thus resulting in more overheads compared to the local/static knowledge-based heuristics. The propagation of delay information can be achieved by executing distributed routing algorithms, such as Bellman–Ford.

A distributed route selection algorithm based on preferred neighbor approach has been proposed in [8]. This paper studied the existing routing heuristics such as SPF and LLF, and found that the SPF performs better in terms of ACAR and ARD under uniform traffic and is poor when the call requests are focused on some hot nodes or links. On the other hand, the LLF tries to balance the load on each link by selecting the preferred neighbor nodes in a round-robin fashion. For unbalanced traffic, it indeed increases the ACAR. But under the uniform traffic, it tries to balance the load on each preferred node. A slightly unbalanced load between two preferred nodes will probably cause the LLF to select a route with a longer distance. Even under light load, the LLF scheme still changes the route dynamically, which is quite unnecessary. To overcome the problems associated with the SPF and LLF, a routing algorithm, called *two-level shortest path* (TSPF), has been proposed in [8]. In this algorithm, the links of a node are grouped into two groups: heavy group and light group, based on a threshold value of load. The SPF heuristic is applied first within the light group and then within the heavy group. These algorithms (SPF, LLF, and TSPF) are poor in terms of ACST since they encounter excessive backtracks under heavy loads.

Two adaptive algorithms, hot-spot avoidance (HSA) routing and virtual-path (VP) routing, have been proposed recently in [2] for massively parallel systems. The HSA algorithm is more suitable for a dynamic environment, whereas the VP algorithm is more suitable for a static environment. Most importantly, these algorithms are inadequate for satisfying QoS parameters, which is typically the requirement in a real-time network.

### C. Motivation for Our Work

It is clear from the preceding discussion that there is no single routing algorithm which is suitable for improving all the three performance metrics under different load conditions. This has motivated us to come up with a routing approach whose objective is to improve all the metrics (ACAR, ACST, and ARD) simultaneously, and cater to different kinds of work loads. As an attempt to satisfy this objective, we propose a distributed routing approach which is a generalization of both the flooding and preferred-neighbor based approaches by offering a wide spectrum of solutions ranging from the one provided by the flooding-based approach to that provided by the preferred-neighbor approach.

## III. New Distributed Route-Selection Approach

In this section, we propose a distributed route-selection approach, called *parallel probing*, which combines the benefits of both the flooding and preferred-neighbor approaches, and also the benefits of multiple preferred-neighbor heuristics that are employed by it in a unified way. In our approach, we search for a qualified path by simultaneously probing at most $k$ different paths using $k$ different heuristics, one for each path. Since searching in parallel (for a qualified path) reduces the number of backtracks as compared to the sequential searching, the ACST is less in the case of parallel search, at the cost

of reserving more resources. For example, when $k$ paths are simultaneously searched by parallel probing, the resources that are tentatively reserved (which are not available for other calls) is approximately $k$ times that of the single path searched by sequential probing.

To alleviate this excessive resource reservation without losing the fast call setup capability, the parallel probing uses a concept called *intermediate destinations* (ID's), which are subset of nodes along the *least-cost* path between the source and destination of a call. The least-cost metric can be in terms of (minimum) number of hops, or based on (least) load, combination of hop count and load, or some other combination. When a call request arrives, the source node $(ID_0)$ first decides the ID's for the call and then initiates probes for a qualified path to the first ID $(ID_1)$ by sending probe packets (the ID list is appended to each probe packet) in parallel on $k$ different paths by employing $k$ different heuristics. The probe packet that first reaches $ID_1$ is considered to be the winner of that segment which originates at the source and ends at $ID_1$. The subsequent probe packets corresponding to the same call reaching at $ID_1$ are rejected and, hence, the resources reserved by them are released immediately up to the previous ID (i.e., the source). Now, the parallel probing starts all over again from $ID_1$ to $ID_2$ in a similar manner. This procedure is repeated until the destination is reached or timeout has occurred. The reverse pass follows the path used during the forward pass, either for relaxing/confirming (when the call is accepted) or for releasing (when the call is rejected) the resources along the path. If the resources reserved at a node are not confirmed within a time interval, then the node automatically releases the resources reserved for that call. When a probe packet is sent from the source, it is assigned a unique identifier, obtained by concatenating the node number with the local request counter and the heuristic identifier. The heuristic identifier indicates the heuristic based on which a node finds its neighbor.

### *A. Node and Procedure Types*

To formally present the parallel-probing approach, we first define some node types, and then define some procedures which will be invoked when a probe packet is received. The nodes of the network, with respect to a call, are classified into four types.

- **SOURCE:** source node of the call.
- **DESTINATION:** destination node of the call.
- **ID:** ID of the call. The call can have more than one ID, and the search for paths proceed from one ID to another. The selection of ID's and the sequence in which the ID's are to be searched is decided by the least-cost metric between the SOURCE and the DESTINATION of the call. Let $ID_0, ID_1, \cdots, ID_{n-1}, ID_n$ be the sequence of ID's of a call. Without loss of generality, $ID_0$ is the SOURCE and $ID_n$ is the DESTINATION.
- **IID:** intended ID (IID) of the call, which is an ID to which route is being probed currently. $ID_i$ is IID iff $ID_{i-1}$ was the previous IID and $ID_{i+1}$ will be the next IID. Without loss of generality, SOURCE $(ID_0)$ is the first IID and DESTINATION $(ID_n)$ will be the last IID.
- **SN:** a simple node (SN). The set of nodes which are not members of any of the above categories. These are nodes between ID's along the least-cost path between SOURCE and DESTINATION.

A *segment* is the path obtained by the parallel-probing algorithm between two consecutive ID's of a call. The procedures executed by nodes during call establishment are classified into six categories: Reserve( ), Forward( ), Release( ), and Backtrack( )—the procedures executed during the forward pass; Relax( ) and Reject( )—the procedures executed during the reverse pass. The ID's of a call are appended to each probe packet, PROBE_PKT, at the source node of the call. Each probe packet also has a path field that captures the sequence of nodes that constitutes the current path.

- **Reserve**$(N, ID_j, PROBE\_PKT)$: Route the probe packet (PROBE_PKT) from node $N$ to its preferred neighbor for reaching node $ID_j$ based on a heuristic. This involves performing a call admission test on the best preferred link of node $N$—checking for the availability of bandwidth required by the call; if the admission test is successful, the resources are reserved. If the admission test fails on the best-preferred link, the call admission test is performed on the next best-preferred link. This is repeated for a fixed number of times. If the call admission is successful, it returns "success," it returns "failure," otherwise.

```
Reserve (N, ID_j, PROBE_PKT)
begin
  Repeat
    If (call admission is successful)
      Reserve bandwidth on the preferred link.
      Send PROBE_PKT to the preferred neighbor.
      return(success).
    Else Select the next preferred link.
  Until (maximum neighbors have been tried).
  return(failure).
end.
```

- **Forward**$(ID_i, IID, PROBE\_PKT)$: This procedure initiates parallel probes on $k$ paths using $k$ different heuristics. For each probe, it invokes a Reserve() procedure. If all probes fail the admission test, then a Release() procedure is invoked to release the resources reserved between $ID_i$ to $ID_{i-1}$.

```
Forward (ID_i, IID, PROBE_PKT)
begin
  Let H_1, H_2, ..., H_k be the k heuristics.
  For p = 1 to k do
    Select the best neighbor based on heuristic H_p.
    status[p] = Reserve(ID_i, IID,PROBE_PKT).
  If (none of status[p] is successful, 1 ≤ p ≤ k) then
    If (ID_i is SOURCE) then call is rejected.
    Else Release(ID_i, ID_{i-1}, PROBE_PKT).
end.
```

- **Release**$(ID_i, ID_{i-1}, PROBE\_PKT)$: Release the resources along the path between ID $ID_i$ and ID $ID_{i-1}$ using the path stored in the PROBE_PKT.

- **Backtrack***(N, PROBE_PKT)*: Backtrack from node $N$ to its predecessor (say $pred(N)$) in the routing path, which is stored in PROBE_PKT. This implicitly releases the resources reserved for the call from $pred(N)$ to $N$. The node $N$ could be of type SN or ID. If $pred(N)$ is SOURCE, the PROBE_PKT is dropped.
- **Relax***(DESTINATION, SOURCE, PROBE_PKT)*: This procedure is invoked when the forward pass is successful. This Relaxes the excess resources such as bandwidth, buffers, and delay guaranteed (delay is an additive metric) along the path, where the path is stored in PROBE_PKT, from DESTINATION to the SOURCE of the call. If cycles are present in the path, they are also removed by releasing the resources reserved in the nodes which form the cycles.
- **Reject***($ID_i$, SOURCE, PROBE_PKT)*: This procedure is invoked when the channel is not possible to set up. This releases the resources along the path, where the path is stored in PROBE_PKT, from node $ID_i$ to SOURCE of the call. The $ID_i$ is the last ID in the ID's list. This means that no ID can become IID to which probe can be initiated. As a special case, $ID_i$ could be the DESTINATION of the call.

### B. Parallel-Probing Algorithm

Fig. 1 shows the pseudo code of the parallel-probing algorithm which is executed when a call request arrives or a probe packet is received at a node. When a probe packet is received, depending on the node type different case statements will be executed.

In parallel probing, at a node of type SN and ID, the resources might be reserved for a call more than once due to multiple probe packets passing through that node. If the call is successful and that node is part of the qualified path, then only one of these reservations will be confirmed by the Relax( ) procedure and the remaining reservations will be released either by Reject( ) or Release( ) procedures.

*1) Properties of Parallel-Probing Approach:* The parallel-probing approach possesses the following properties:

- **Liveness:** The use of ID's ensures the forward movement of call probe packets towards the destination. This eliminates the possibility of a probe packet getting stuck within a group of nodes that form a cycle.
- **Adaptiveness:** The different segments of a qualified path could very well be selected by different heuristics employed (as shown in Fig. 2) depending on the load condition on that segment.
- **Generality:** The parallel-probing approach reduces to the flooding approach if the $k$ heuristics selected for parallel probing are the same, and it reduces to the preferred neighbor approach when the number of paths searched in parallel is one, i.e., $k = 1$.
- **Cycle-Free Path:** The qualified routing path produced by the parallel probing is cycle free.

*Theorem 1:* Any qualified routing path produced by the parallel probing is cycle free.

*Proof:* There are two types of cycles possible: 1) cycles involving SN's, i.e., the originating and terminating nodes of a cycle is an SN and 2) cycles involving ID's, i.e., the originating and terminating nodes of a cycle is an ID.

The first type of cycles cannot exist in a path produced by the parallel-probing algorithm, since the heuristics employed for parallel probes ensure cycle freeness in each segment of the path. Even if the heuristics allow cycles, such cycles can be removed during the reverse pass. The second type of cycles could arise in two ways as discussed below.

The first possibility is when IID is $ID_k$ and the current node is $ID_i$, such that $i<(k-1)$. This means that the probe packet has reached an ID which had become IID in the past, which implies a cycle of the form $ID_0, ID_1, \cdots, ID_i, \cdots, ID_{k-1}, \cdots, ID_i$. The portion of path from $ID_{k-1}$ to $ID_i$ has to be removed. The first step of case ID of the algorithm exactly does this.

The second possibility arises when the following sequence of events takes place.

a) The IID is $ID_k$ and the current node is $ID_i$ such that $i>k$; This means that a probe packet has reached an ID which will become IID in the future; In such cases, the ID node forward the probe packet just like an SN node (the last Else part of case ID of the algorithm).
b) $ID_i$ becomes IID. The sequence of events (a) and (b) results in creation of a cycle of the form $ID_0, ID_1, \cdots, ID_i, \cdots, ID_k, \cdots, ID_i$. The portion of the path from $ID_i$ to $ID_k$ to $ID_i$ has to be removed. Such types of cycles are removed during reverse pass of the algorithm either by the Relax( ) procedure or by the Reject( ) procedure.

*Theorem 2: Dangling resources* are unusable resources that are reserved on some links (and nodes) such that they are neither part of any successfully established real-time channel nor part of any ongoing channel-setup attempt. The parallel-probing algorithm does not leave dangling resources in the network.

*Proof:* To prove this, we identify the procedures which reserve resources and which release resources and also the node types which execute these procedures. Resource reservation is done by procedures Reserve( ) and Forward( ), and either or both of these are executed by nodes of type SOURCE, IID, ID, and SN. Reserve( ) reserves on a single link and Forward() reserves on $k$ links. Resource release is done by procedures Release( ) and Backtrack( ), and either or both of these are executed by nodes of type IID, ID, SN, and DESTINATION. The forms of unnecessary resource reservations, which would become dangling, and the proper release of such resources are listed below.

- Since the algorithm ensures the release of resources between IID (say $ID_i$) and $ID_{i-1}$ for all the nonfirst probe packets reaching IID, by executing Release( ), there are no dangling resources between two consecutive ID's.
- When an ID, say $ID_i$, becomes the last element in the ID's list, the resources reserved from $ID_i$ to SOURCE are released by Reject( ). As a special case, $ID_i$ could be the DESTINATION.
- The algorithm detects some cycles, involving ID's, in the forward pass and removes the unnecessary resources in

**Parallel Probing(SOURCE, DESTINATION, $k$)**

**begin**

**1.** When a new call request arrives do the following.

SOURCE:

Assemble probe packet (PROBE_PKT) with intermediate destinations $(ID_0, ID_1, ..., ID_{n-1}, ID_n)$, without loss of generality, $ID_0$ is SOURCE and $ID_n$ is DESTINATION

$IID = ID_1$; Forward(SOURCE, $IID$, PROBE_PKT) /* Initiates parallel probing on $k$ paths */

**2.** When a probe packet arrives do the following.

**switch** (current node type) /* node type is with respect to the probe packet */

**case** IID: Let the current node be $ID_i$. /* Intended intermediate destination */

**If** (PROBE_PKT is already seen by $ID_i$) **then** Release($ID_i$, $ID_{i-1}$, PROBE_PKT). /* for loser packet */

**Else** $IID = ID_{i+1}$; Forward($ID_i$, $IID$, PROBE_PKT). /* for winner packet - the first PROBE_PKT */

**case** ID: Let the current node be $ID_i$. /* Intermediate destination */

**If** (IID is $ID_k$ such that $i < (k-1)$) **then** /* reached a past ID; cycle is encountered */

Release($ID_i$, $ID_{k-1}$, PROBE_PKT). /* cycle removal */

**Else If** (IID is $ID_{i+1}$ and all the $k$ probe packets have returned) **then** /* IID is not reachable */

Delete $ID_{i+1}$ from IDs list. /* this is to keep track of the actual IDs list */

**If** ($ID_i$ is the last element in the IDs list) **then**

Reject($ID_i$,SOURCE, PROBE_PKT). /* call is rejected. */

**Else** $IID = ID_{i+2}$; Forward($ID_i$, $IID$, PROBE_PKT). /* parallel probe to new IID */

**Else**

status = Reserve($ID_i$, IID, PROBE_PKT). /* already known IID */

**If** (status is failure) **then** Backtrack($ID_i$, PROBE_PKT).

**case** SN: Let the current node be $N$. /* Simple node */

status = Reserve($N$, IID, PROBE_PKT). /* already known IID */

**If** (status is failure) **then** Backtrack($N$, PROBE_PKT).

**case** DESTINATION:

**If** (PROBE_PKT is already seen) **then** /* for loser packet */

Release(DESTINATION, $ID_{n-1}$, PROBE_PKT).

**Else if** (call is acceptable) **then** /* for winner packet and call is acceptable */

Analyse for cycles - mark the nodes in the path that form the cycles.

Relax(DESTINATION, SOURCE, PROBE_PKT). /* call is accepted - reverse pass */

**Else** Reject(DESTINATION, SOURCE, PROBE_PKT). /* call is rejected - reverse pass */

**end.**

Fig. 1. Parallel-probing algorithm.

the cycles (refer case ID of the algorithm). Similarly, it removes other cycles during reverse pass by Relax( ).

- When Backtrack( ) takes place at an ID or SN, the resources reserved on a link are released.

Thus, all the reserved resources are either used by the successful channels or properly released or part of ongoing channel-setup attempts.

*Theorem 3:* A call setup initiated at a node is either set up or rejected in a finite time.

*Proof:* A call setup is initiated by executing Forward( ) at the SOURCE node of the call. Since the algorithm has liveness property, i.e., forward movement of probe packets from one ID to another in the order of $ID_0$, $ID_1, \cdots, ID_{n-1}, ID_n$, it completes the forward pass in a finite time. It is obvious that the reverse pass also takes finite time by executing either Relax( ) (for successful setup) or Reject( ) (for call rejection) procedure.

### C. Example for Parallel Probing

Fig. 2 depicts how the parallel-probing algorithm establishes a channel. Here $k = 2$. Note that the established path has two segments, in which the first segment is selected by heuristic H1 and the second is by heuristic H2. The sequence of events taking place in Fig. 2 is given below.

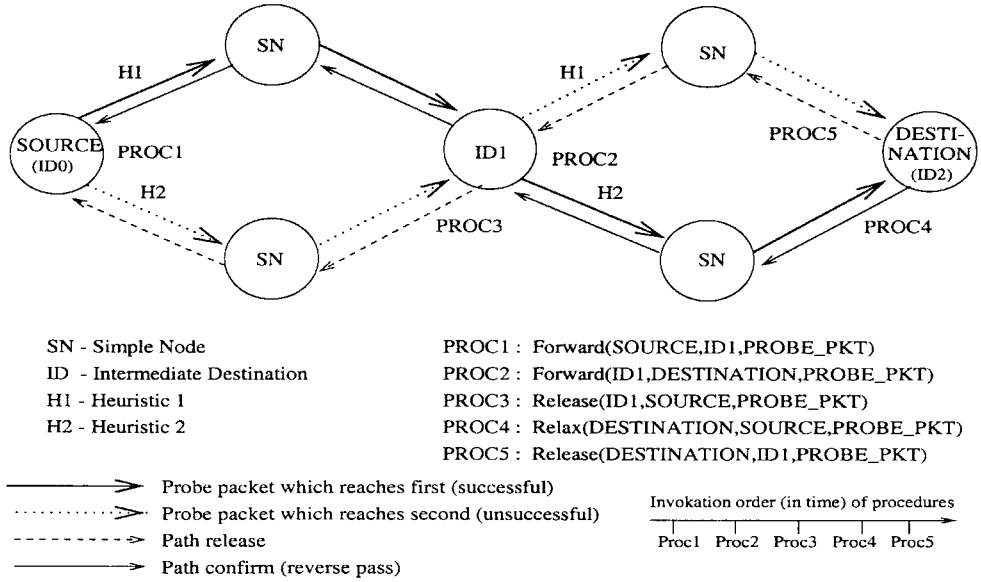



Fig. 2. Typical channel establishment by parallel probing.

1) To start with IID $= ID_1$. SOURCE executes procedure Forward(SOURCE,IID,PROBE_PKT). This means sending two probe packets, one using heuristic H1 (on upper link) and the other using heuristic H2 (on lower link).
2) Probe packet corresponding to heuristic H1 reaches $ID_1$ (which is the current IID) first. (This is the winner packet.)
3) Now, IID = DESTINATION. $ID_1$ executes procedure Forward($ID_1$,DESTINATION,PROBE_PKT). This means sending two probe packets, one using H1 (on upper link) and the other using H2 (on lower link).
4) Heuristic H2's probe packet, which was sent from SOURCE, reaches $ID_1$. (This is a loser packet.)
5) $ID_1$ executes procedure Release($ID_1$,SOURCE, PROBE_PKT). This is to immediately release the resources in the segment between SOURCE and $ID_1$ obtained using heuristic H2.
6) DESTINATION (which is the current IID) receives heuristic H2's probe packet which was sent from $ID_1$. (This is the winner packet).
7) DESTINATION accepts the call and executes procedure Relax(DESTINATION,SOURCE,PROBE_PKT). This is to confirm the channel from the DESTINATION to the SOURCE.
8) DESTINATION receives heuristic H1's probe packet which was sent from $ID_1$. (This is a loser packet.)
9) DESTINATION executes procedure Release(DESTINATION,$ID_1$,PROBE_PKT). This is to release the resources in the segment between $ID_1$ and DESTINATION obtained using heuristic H1.

## IV. Simulation Studies

We have conducted extensive simulation studies to evaluate the performance of our parallel-probing approach in terms of ACAR, ACST, and ARD for a wide range of traffic and QoS parameters for different network topologies. Before presenting the results, we describe the simulation model and the parameters used in the simulation.

### A. Simulation Model

To study the effectiveness of the proposed parallel-probing approach, in terms of all the three metrics, we have compared its performance with that of the flooding algorithm [15] and the TSPF algorithm [8]. Since the objective of any routing algorithm is to find a qualified path with minimal operational overheads, we chose local/static knowledge based heuristics for parallel probes. The probe packets are assumed to use control channels for path search. The number $(k)$ of parallel paths searched simultaneously by the parallel-probing approach is taken as two, and the heuristics employed are SPF and LLF.

The ID's for the parallel-probing approach are chosen based on the shortest path between the source and destination of the call. For example, let the length of the shortest path be $n$ and the number of ID's be $m$ excluding source and destination. Then, the $ID_i$ of a call is the $(i * (n/m + 1))$th node in the shortest path. In our simulation, flooding algorithm sends a message to all its neighbors except the node from which the packet has come. The call admission test used, at each node, is the admission test of the Hierarchical Round Robin (HRR) [18], [19] scheduling algorithm. HRR is a rate-based scheduling discipline which has a simple admission test: to admit a new channel on a link, it checks whether the sum of the utilization (utilization of a channel is the ratio of its maximum message length to the period of its frame size) of all the channels passing through the link (including the new channel) is less than, or equal to, one.

Two different network topologies have been considered for evaluating the performance of the different route selection algorithms. For performance study in a wide area network, the ARPA network shown in Fig. 3 (21 nodes, 26 links) and the

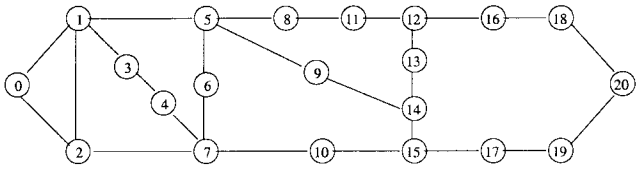



Fig. 3. The ARPA network (21 nodes, 26 links).

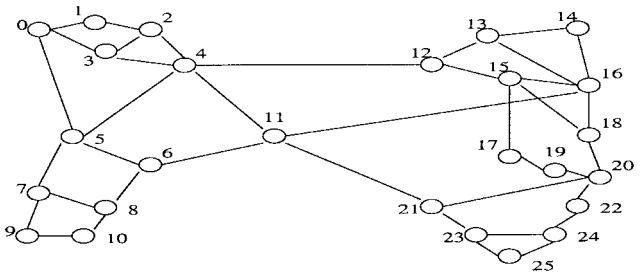



Fig. 4. The USA network (26 nodes, 39 links).

USA network shown in Fig. 4 (26 nodes, 39 links) have been taken as representative topologies [8]. In our simulation, the delay and bandwidth of the links of the networks are taken to be 1 and 100, respectively. Each point in the simulation curves is an average of five simulation runs. For each simulation run, 5000 call requests are generated. The simulation parameters are given in Fig. 5. The call requests are generated according to the following two distributions.

1) **Uniform distribution:** The source-destination pair of a call is uniformly chosen from the node set.
2) **Hot communication pair (HCP) distribution:** $p\%$ of all call requests are set to a particular hot source and destination pair, the other calls follow uniform distribution.

The other parameters of a call are generated as follows.

- The duration, bandwidth, and end-to-end delay requirements of a call are uniformly distributed between their respective minimum and maximum values.
- The inter-arrival time of call establishment requests follow exponential distribution with mean $1/\lambda$ for each node.

For studies in Figs. 6–10 and 12, there is no hot communication pair, and for studies in Figs. 6–11, the number of ID's used by the parallel-probing approach is taken as two. To recall, the main objective of this paper is to propose a route selection approach which attempts to improve all the three performance metrics simultaneously, since all of them are important in a real-time network, and also to adapt to different load conditions. To this effect, the experiments are so designed to study the effect of different QoS and traffic parameters for both uniform and hot-pair (nonuniform) communication patterns. Here, we present the results for the ARPA network (Figs. 6–12) only. The results for the USA network exhibit similar behavior that of the ARPA network.

### *B. Simulation Results*

From Figs. 6(a)–11(a), it can be observed that the parallel-probing approach offers higher ACAR than the other two algorithms for all the parameters varied. This is due to less call setup overhead (in terms of resource reservation while path probing), due to the use of ID's, and high adaptiveness of the parallel probing, in terms of using multiple heuristics for path probing, i.e., it allows a routing path to have different segments selected by different heuristics, whereas the flooding suffers due to tentative reservation of resources on multiple paths simultaneously, which increases blocking of new calls. On the other hand, the TSPF suffers due to its less adaptiveness, and is still better than the flooding.

| parameter | explanation | value taken when | |
|---|---|---|---|
| | | varied | fixed |
| $\lambda$ | call arrival rate | 0.6,0.7,...,1.0 | [0.5,0.65] |
| *del-min* | minimum end-to-end delay of a call | - | 10 |
| *del-max* | maximum end-to-end delay of a call | 10,20,...,50 | 20 |
| *bw-min* | minimum bandwidth of a call | - | 2 |
| *bw-max* | maximum bandwidth of a call | 2,4,...,12 | 4 |
| *dur-min* | minimum duration of a call | - | 200 |
| *dur-max* | maximum duration of a call | 200,400,...,1200 | 400 |
| *num-ID* | number of intermediate destinations excluding source and destination | 0,1,2,...,5 | 2 |
| *hp-pair* | % of hot-pair communication | 10,20,...,60 | 0 |
| *stime* | maximum call setup time | 3,4,...,7 | - |

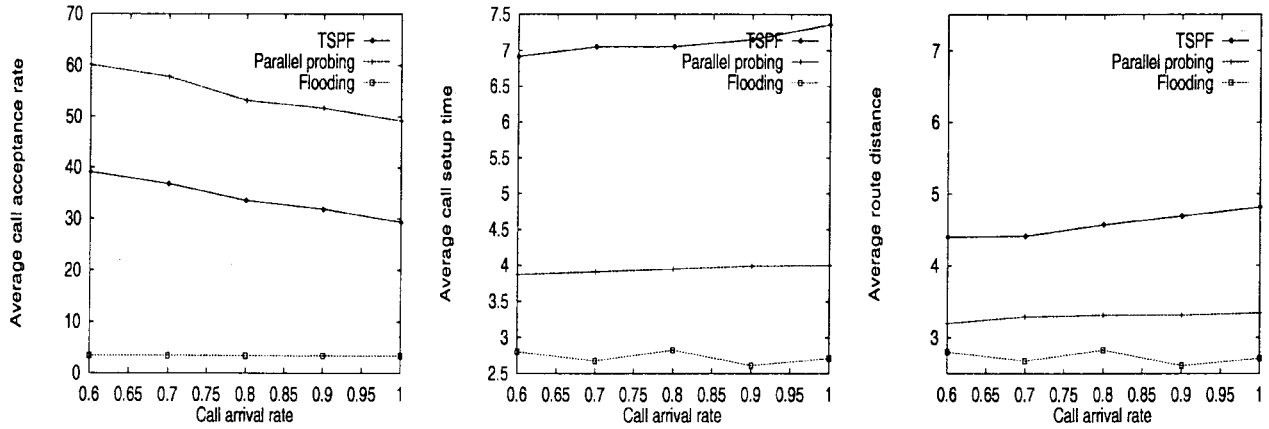
Fig. 5. Simulation parameters.

From Figs. 6(b)–11(b) and 6(c)–11(c), it can be seen that the ACST and ARD offered by the flooding are smaller than that of the other two algorithms. This is due to simultaneous probing for paths on all links of a node. Also, note that the ACST and ARD for the flooding are the same because it does not encounter backtrack. The ACST and ARD of the parallel-probing approach are very close to that of flooding and better than that of TSPF due to its controlled backtracking nature.

*1) Effect of Call Traffic Characteristics:* The effect of traffic characteristics of a call, namely, call arrival rate, call setup time, and call duration, are studied in Figs. 6–8, respectively.

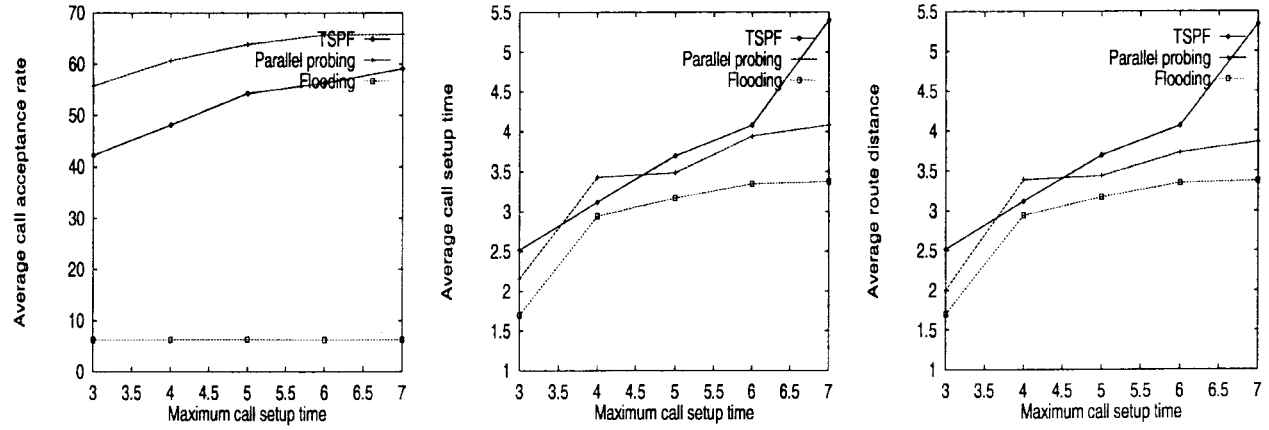
From Fig. 6(a), the ACAR decreases for increasing values of call arrival rate. This is because of more calls arrive at higher loads which, in turn, consume more resources, thus resulting in more calls getting dropped. That is, at higher loads, more and more calls attempt to use the fixed available resources. Also, it can be observed that the ACST and ARD increase with load. This is because at higher loads, finding a qualified path involves more search in the network as most of the resources are already in use by the existing channels.

In Fig. 7, the effect of varying call setup time is studied. The increase in call setup time introduces a trade-off between improvement in ACAR due to more backtracks and deterioration in ACAR due to excess resource reservation for a longer time. From Fig. 7(a), as the call setup time increases, the ACAR improvement phenomenon is effective for all the three algorithms (not very significant for flooding). This is because the route selection algorithms are allowed to probe for channels for a longer time (more backtracks), which increases the chances of establishing a channel. Note that, the ACAR curve corresponding to flooding is almost flat since flooding does not backtrack. Also, note that, as the call setup time increases, the ARD also increases for all the three algorithms. This is also because the algorithms are allowed to probe for paths for longer times resulting in longer paths.



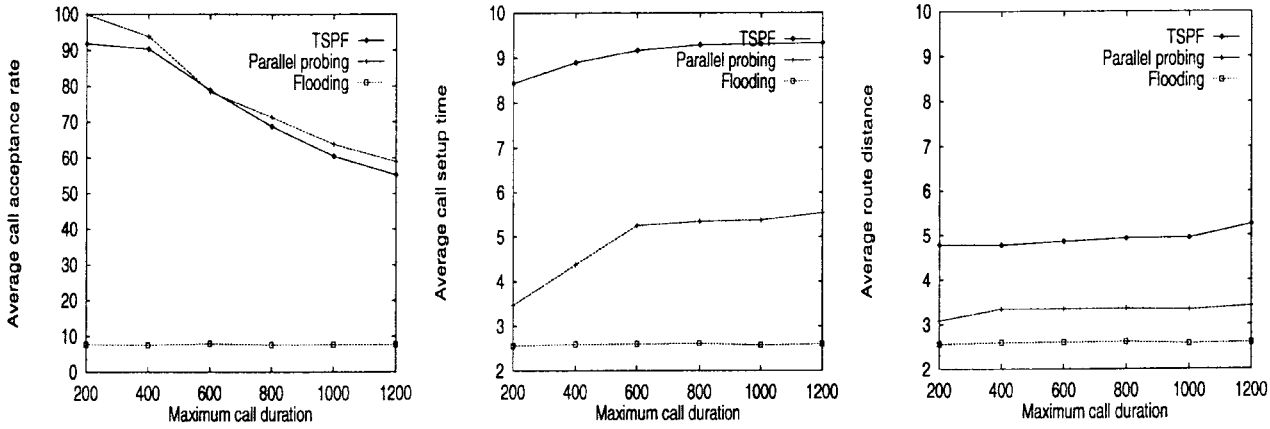

(a) (b) (c)

Fig. 6. Effect of call arrival rate on (a) ACAR, (b) ACST, and (c) ARD.




(a) (b) (c)

Fig. 7. Effect of call setup time on (a) ACAR, (b) ACST, and (c) ARD.

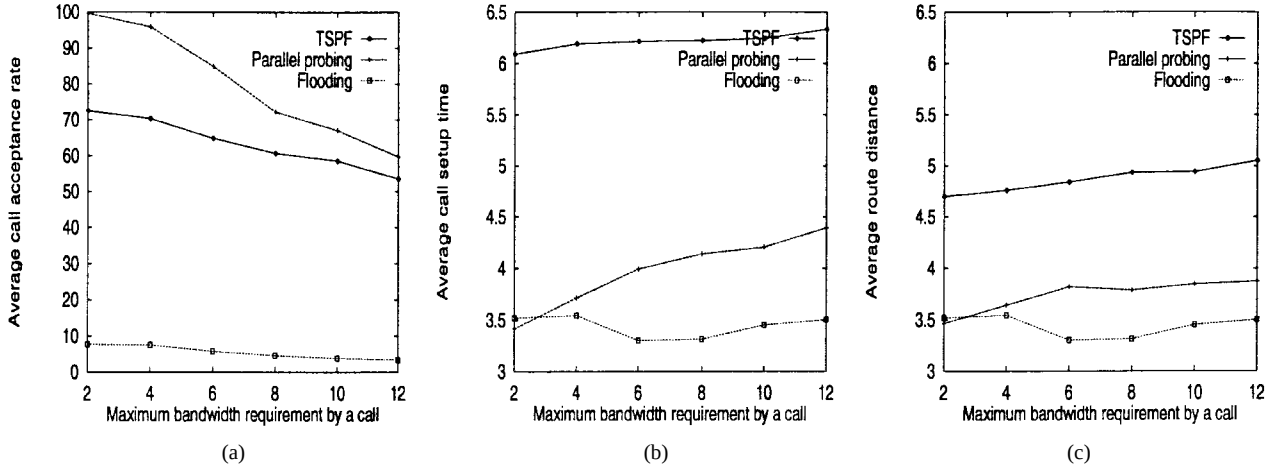



(a) (b) (c)

Fig. 8. Effect of call duration on (a) ACAR, (b) ACST, and (c) ARD.

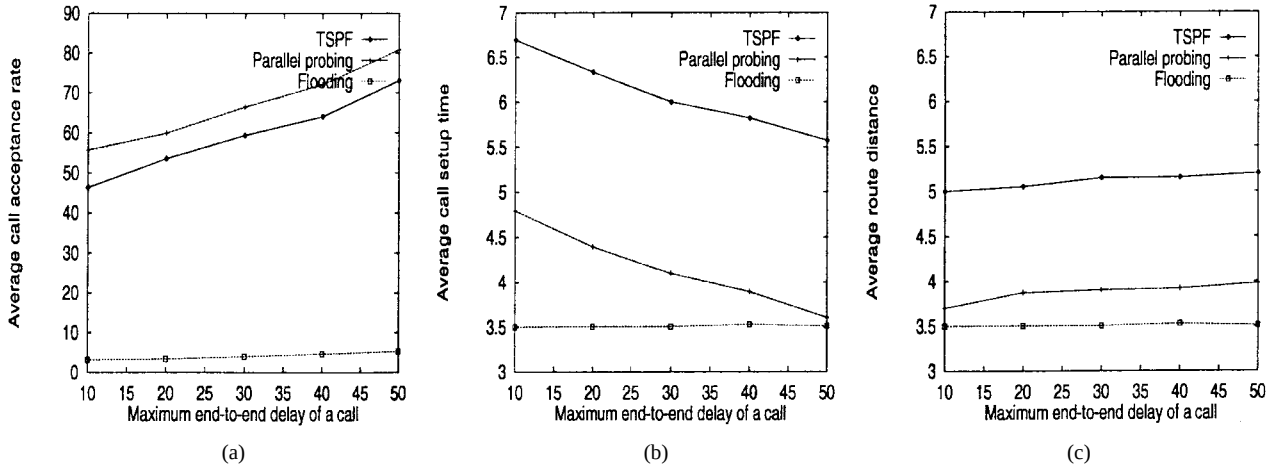
The ACAR deterioration phenomenon will be prominent when the ratio of call duration to call setup time is much less. In our experiments, we have chosen this ratio to be approximately 100 ($\approx$ 300/3.5), which is a large value and hence the excess resource reservation effect is outperformed by gain obtained by more backtracks.

From Fig. 8(a), it can be observed that the ACAR decreases with increasing call duration. This is because for higher values of call duration, the resources are blocked (not available for the new calls) by the currently active calls for longer durations. From Fig. 8(b) and (c), the increasing call duration increases the call setup time for parallel probing and TSPF because




(a) (b) (c)

Fig. 9. Effect of maximum bandwidth requirement by a call on (a) ACAR, (b) ACST, and (c) ARD.




(a) (b) (c)

Fig. 10. Effect of maximum end-to-end delay of a call on (a) ACAR, (b) ACST, and (c) ARD.

of blocking of resources. The setup time of flooding does not increase much because of its flooding (and hence no backtracking) nature.

*2) Effect of Call QoS Requirements:* The effect of QoS requirements of a call, namely, the bandwidth and end-to-end delay requirements are studied in Figs. 9 and 10, respectively. From Fig. 9(a), the ACAR offered by all the three algorithms starts decreasing for increasing values of bandwidth requirement. The reason is due to reserving of more bandwidth for the currently active calls, when call bandwidth is more, which results in blocking of new calls. Also for the same reason, the ACST and ARD increase with increasing bandwidth requirement.

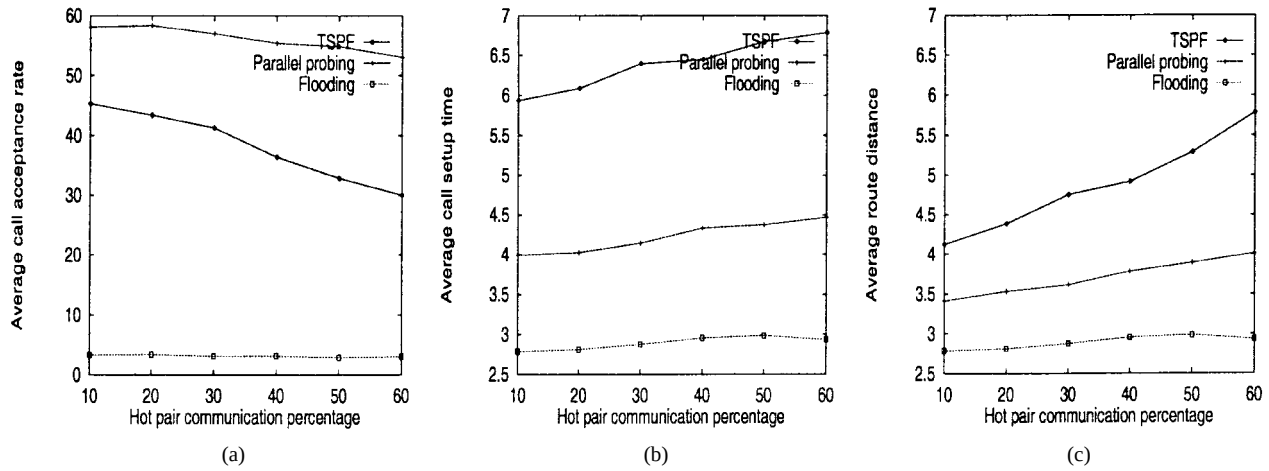
The effect of varying end-to-end delay constraint imposed by a call is studied in Fig. 10. As the end-to-end delay increases, the ACAR also increases. This is because the chances of meeting the delay constraints of a call is more when the end-to-end delay is large, i.e., the nodes along the path of a call can assign a higher node deadline (delay) for the messages of the call. The ACST metric decreases with increasing end-to-end delay for the same reason.

*3) Effect of Hot Pair Communication:* For this study, nodes 3 and 14 of the ARPA network (Fig. 3) are chosen as Hot Communication Pair (HCP). From Fig. 11, the ACAR decreases with increasing HCP, and the ACST and ARD increase with increasing hot communication percentage. This is due to the fact that the increase in hot communication saturates the paths between the HCP nodes after reserving resources for some number of calls, and hence rejects the subsequent calls until some of the resources are released due to call tear down. This is applicable to all the three algorithms.

In conclusion, the parallel-probing approach offers higher call acceptance rate than that of the TSPF and flooding, and its setup time and routing distance are closer to that of the flooding. The parallel probing is better than flooding and TSPF in improving all the three metrics simultaneously for different simulated load conditions. This conclusion is applicable for both the APRA and USA network topologies.

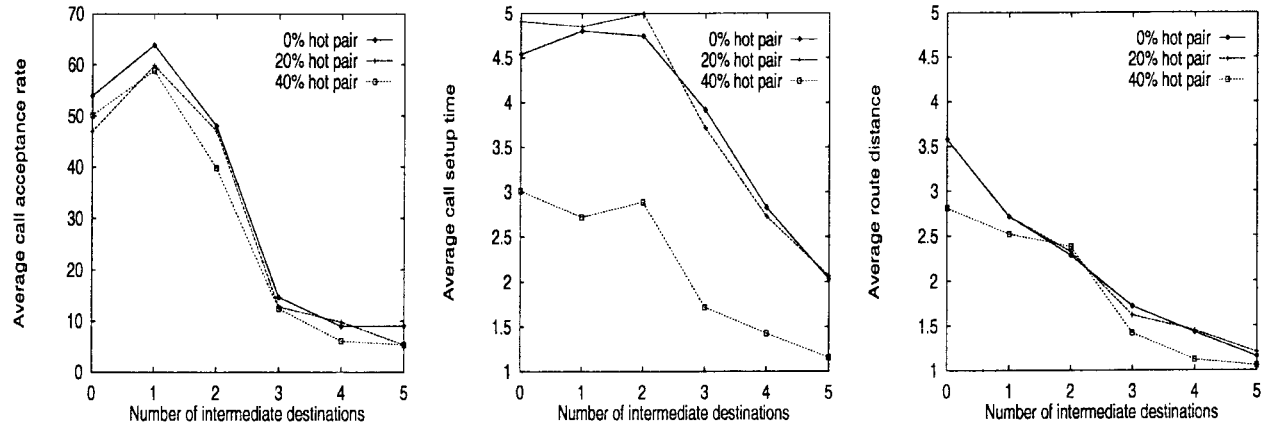
*4) Effect of Number of ID's:* The effect of number of ID's used in the parallel-probing approach is studied in Fig. 12. For this, 0%, 20%, and 40% HCP is considered; nodes 3 and 14 of the ARPA network are chosen as HCP.

Note that the concept of routing to ID's has been introduced with the purpose of controlling the excessive resource reservation on parallel paths. The choice of number of ID's is very crucial in deciding the overall performance of the network.




(a) (b) (c)

Fig. 11. Effect of hot pair communication on (a) ACAR, (b) ACST, and (c) ARD.




(a) (b) (c)

Fig. 12. Effect of number of ID's on (a) ACAR, (b) ACST, and (c) ARD.

When the number of ID's is very low, excessive resource reservations along the path occur, which in turn reduces the ACAR. As a special case, when *num-ID* is zero, the parallel probing reduces to $k$-path flooding, which reserves resources on $k$ paths from source up to the destination (in the worst case) during forward pass. On the other hand, when the number of ID's is very large, the routing path is dictated by the least-cost metric which is used to identify the ID's. As a special case, when *num-ID* is equal to the length of the least-cost path, then the parallel probing reduces to the least-cost path heuristic. The proper choice of *num-ID* depends on the topology of the network. When the topology is dense, higher value of *num-ID* is preferable in order to reduce the excessive reservation by probe packets which arises due to the existence of more disjoint paths between any source-destination pair. Any reasonable value of *num-ID* should be much less than the diameter of the network.

From Fig. 12 (for ARPA), the overall peak performance (by simultaneously considering all the three metrics) occurs when the number of ID's (*num-ID*) is one. The peak performance for the USA network topology was found to occur when *num-ID* is 2, with nodes 3 and 19 chosen to be HCP.

Similarly, the choice of $k$ is also crucial in deciding the overall performance of the network. When the topology is sparse, a lower value of $k$ is preferable, because a higher value of $k$ will make the probe packets to visit many common nodes and, hence, reserving resources multiple times in such nodes, thereby reducing ACAR. In parallel probing, apart from performance point of view, extra processing overhead comes in the form of sending $k$ probe packets at every ID, as opposed to one probe packet in conventional preferred-neighbor algorithms such as SPF, LLF, and TSPF. This extra overhead would be much less, as the number of ID's is very less compared to the diameter of the network.

## V. Conclusion

In this paper, we have proposed a new distributed routing approach, called *parallel probing*, for real-time channel establishment in a point-to-point network, which is a generalization of preferred-neighbor based and flooding-based routing approaches. The parallel probing attempts to improve all three performance metrics (call acceptance rate, call setup time, call routing distance) simultaneously by combining the benefits of preferred-neighbor approach (better call acceptance) and flooding-based approach (minimum call setup time and routing distance). Further, it has the flexibility of accommodating any routing heuristic as one of the heuristics for parallel search in order to adapt to different load conditions. We

have demonstrated the effectiveness of the proposed parallel-probing approach through simulation for a wide variety of QoS and traffic parameters for different load conditions (uniform and hot-pair communication) for two well known network topologies. Our simulation studies reveal the following.

- The parallel probing always offers higher call-acceptance rate than one of the known preferred-neighbor algorithms, such as the two-level shortest path algorithm.
- The call setup time and route distance offered by the parallel probing are closer to the flooding, and are always lower (better) than that of the two-level shortest path algorithm.
- The proper values of number of ID's and $k$ are crucial in deciding the performance of the parallel probing, and they depend on the connectivity of the network.

In conclusion, the parallel-probing approach is better in terms of improving multiple performance metrics simultaneously for a wide range of QoS and traffic parameters for diverse communication (uniform and hot-pair) traffic. Currently, we are extending the parallel-probing approach to the multicast routing.

## ACKNOWLEDGMENT

The authors would like to thank R. Sriram and anonymous reviewers for their suggestions to improve the presentation of the paper.

## REFERENCES

[1] C. M. Aras, J. F. Kurose, D. S. Reeves, and H. Schulzrinne, "Real-time communication in packet-switched networks," *Proc. IEEE*, vol. 82, pp. 122–139, Jan. 1994.
[2] M. Boari, A. Corradi, and C. Stefanelli, "Adaptive routing for dynamic applications in massively parallel architectures," *IEEE Parallel Distrib. Technol.*, Spring 1995, pp. 61–74.
[3] S. Chen and K. Nahrstedt, "An overview of quality of service routing for next-generation high-speed networks: Problems and solutions," *IEEE Network*, pp. 64–79, Nov./Dec. 1998.
[4] S. Chen and K. Nahrstedt, "Distributed quality-of-service routing in high-speed networks based on selective probing," in *Proc. IEEE Local Computer Networks (LCN)*, 1998, pp. 80–89.
[5] _____, "Distributed QoS routing with imprecise state information," in *Proc. IEEE Int. Conf. Computer Communications and Networks*, 1998, pp. 614–621.
[6] M. R. Garey and D.S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness.* San Francisco, CA: Freeman, 1979.
[7] D. Ferrari and D. C. Verma, "A scheme for real-time channel establishment in wide-area networks," *IEEE J. Select. Areas Commun.*, vol. 8, pp. 368–379, Apr. 1990.
[8] N. Haung, C. Wu, and Y. Wu, "Some routing problems on broadband ISDN," *Comput. Networks and ISDN Syst.*, no. 17, pp. 101–116, 1994.
[9] D. D. Kandlur, K. G. Shin, and D. Ferrari, "Real-time communication in multihop networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 5, pp. 1044–1056, Oct. 1994.
[10] W. C. Lee, M. G. Hluchyi, and P. A. Humblet, "Routing subject to quality of service constraints in integrated communication networks," *IEEE Network*, July/Aug. 1995.
[11] P. Ramanathan and K. G. Shin "Delivery of time-critical messages using a multiple copy approach," *ACM Trans. Compu. Syst.*, vol. 10, no. 2, pp. 144–166, May 1992.
[12] S. Rampal, D. S. Reeves, and D. P. Agrawal, "An evaluation of routing and admission control algorithms for real-time traffic in packet-switched networks," in *Proc. High Performance Networking*, 1994, pp. 77–91.
[13] H. F. Salama, D. S. Reeves, and Y. Viniotis, "A distributed algorithm for delay-constrained unicast routing," in *Proc. IEEE INFOCOM*, 1997, pp. 84–91.
[14] S. Shenker and L. Breslau, "Two issues in reservation establishment," in *Proc. ACM SIGCOMM*, 1995, pp. 14–26.
[15] K. G. Shin and C. Chou, "A distributed route-selection scheme for establishing real-time channels," in *Proc. High Performance Networking*, 1995, pp. 319–330.
[16] R. Sriram, G. Manimaran, and C. Siva Ram Murthy, "Preferred link based delay-constrained least cost routing in wide area networks," *Comput. Commun.*, vol. 21, no. 18, pp. 1655–1669, Nov. 1998.
[17] Z. Wang and J. Crowcroft, "Quality-of-service routing for supporting multimedia applications," *IEEE J. Select. Areas Commun.*, vol. 14, pp. 1228–1234, Sept. 1996.
[18] H. Zhang and S. Keshav, "Comparison of rate-based service disciplines," in *Proc. ACM SIGCOMM*, Sept. 1991. pp. 113–121.
[19] H. Zhang, "Service disciplines for guaranteed performance service in packet-switching networks," *Proc. IEEE*, vol. 83, pp. 1374–1396, Oct. 1995.

**G. Manimaran** (M'99) for biography, see p. 529 of the August 1999 issue of this TRANSACTIONS.

**Hariharan Shankar Rahul** received the B.Tech. degree in computer science and engineering from the Indian Institute of Technology, Madras, India, in 1997, and the S.M. degree in computer science from the Massachusetts Institute of Technology (MIT), Cambridge, in 1999.

As a research assistant with the Intelligent Networks and Adaptive Transmission Group, Laboratory for Computer Science, MIT, he has worked and published in areas related to the design of congestion control schemes and adaptive applications for the Internet. He has also been a Teaching Assistant for courses in computer systems.

Mr. Rahul is a recipient of the President of India Gold Medal from the Indian Institute of Technology, and the AT&T Leadership Award.

**C. Siva Ram Murthy** (M'98), for biography, see p. 529 of the August 1999 issue of this TRANSACTIONS.