ELSEVIER

Computer Networks and ISDN Systems 30 (1998) 1029–1043

COMPUTER NETWORKS and ISDN SYSTEMS

Decentralized network connection preemption algorithms

Mohammad Peyravian [a,1], Ajay D. Kshemkalyani [b,2]

[a] *IBM Corporation, P.O. Box 12195, Research Triangle Park, NC 27709, USA*

[b] *ECECS Department, P.O. Box 210030, University of Cincinnati, Cincinnati, OH 45221-0030, USA*

Abstract

Connection preemption provides available and reliable services to high-priority connections when a network is heavily loaded and connection request arrival patterns are unknown, or when the network experiences link or node failures. Coupled with the capability to reroute connections (preempted due to failure or preemption), connection preemption allows a high quality of service to be provided to network connections and bandwidth to be used more efficiently. The main contributions of this paper are the following. It presents a comprehensive simulation study of preemption in a general connection-oriented network setting. Our simulation study also provides useful insights into connection preemption and network dimensioning problems in order to achieve a desired level of network availability. Based on the observations made in this study, we designed two connection preemption selection algorithms that operate in a decentralized/distributed network where individual link managers run the algorithm for connection preemption selection on their outgoing links. The first algorithm optimizes the criteria of (i) the bandwidth to be preempted, (ii) the priority of connections to be preempted, and (iii) the number of connections to be preempted, in that order, and has exponential complexity. The second algorithm optimizes the criteria of (i) the number of connections to be preempted, (ii) the bandwidth to be preempted, and (iii) the priority of connections to be preempted, in that order, and has polynomial complexity. From a comparison study of these two algorithms we conclude that the polynomial algorithm is almost as good as the exponential algorithm in terms of overall network performance. © 1998 Elsevier Science B.V. All rights reserved.

Keywords: Network; Preemption; Algorithm; Optimal; Bandwidth; Connection; Priority; ATM; Scheduling

1. Introduction

Nonstationary network conditions may occur for various reasons such as failure of links or nodes, high traffic, and a priori unknown traffic patterns. With the increasing dependence on connection-oriented communications networks such as Asynchronous Transfer Mode (ATM), it is becoming increasingly important to provide not only acceptable steady state performance but also reasonably good performance under nonstationary conditions when demands for network resources are significantly higher [10]. Under nonstationary conditions, if all existing and new connection requests cannot be accommodated, the only possible solution is to preempt certain connections. When preemption becomes inevitable during nonstationary conditions, the preemption policy must minimize the impacts on connections with greater ‘‘value’’ at the cost of possibly

[1] E-mail: peyravn@vnet.ibm.com.
[2] E-mail: ajayk@ececs.uc.edu.

increased impacts on connections with lower ''value''. Connection preemption can also be used as a mechanism for bandwidth reservation and management.

The importance or value of a connection, which can also relate to the connection's quality of service (QoS) requirements as in [7,10], can be expressed by a priority level. The priority levels can be preassigned by the end-system or by the network administrator using various factors such as reliability desired, pricing structure, bandwidth requirement, real-time delivery constraints, desired blocking probability, and nature of traffic such as multimedia, voice, and facsimiles.

To minimize the impacts of nonstationary network conditions on high-priority connections, it might be necessary for new or rerouted high-priority connections to be able to preempt ongoing connections of lower priorities. Preemption makes available bandwidth for new or rerouted high-priority connections, allowing them to proceed. A preempted connection may have to be rerouted, which in turn can cause other ongoing connections of even lower priority to be preempted – this situation occurs when connection preemption is coupled with the capability to reroute connections. When a connection is rerouted, the reroute can be successful or the connection can be dropped.

When connection preemption is inevitable, an algorithm has to choose one or more ongoing connections of lower priorities for preemption in order to establish the high-priority connection that triggered the preemption algorithm. The algorithm must be such that it causes a minimum of disruption in the network due to the preempted connections. In addition, this algorithm must be fast to minimize the duration of disruption or the connection setup time of a preempted connection rerouted due to a failure or unavailable bandwidth. Therefore, the algorithm must be a real-time algorithm [5].

Voice and video connections are some examples of traffic that benefit from preemption. Even if a connection is required to be reliable at the application layer, e.g., TCP, it can still be preempted and rerouted as packets lost during the preemption and rerouting will be retransmitted at the application layer. Thus, TCP can run over the preemption service.

1.1. Previous work

Garay and Gopal [2] addressed the connection preemption selection problem in a centralized network environment and showed that the problem of selecting which connections to preempt in order to minimize the number of connections to be preempted or to minimize the amount of bandwidth to be preempted is NP-complete. Knowing the computational intractability of the problem, they presented a set of heuristic connection preemption selection algorithms. Their algorithms are suitable for a centralized network environment wherein a central control point performs most of the network control functions because information about the complete route of the preempting connection as well as the complete routes of the connections that share one or more links with the preempting connection is required as input to the algorithm. Therefore, a designated entity monitors information about the whole network, i.e., the complete route of the preempting connection as well as the routes of the connections that share one or more links with the preempting connection, and runs the preemption algorithm to select the connections for preemption. Due to the inherent nature of a distributed system, it follows that the designated centralized entity will not have the most up-to-date information about the network. Moreover, when a link or node failure triggers preemption, it is not desirable to have a centralized control point consider the end-to-end path for connection preemption candidates, and a distributed scheme is preferred for scalablity and efficiency.

The upper bound on the computational complexity of their heuristic algorithms is $O(n \cdot m^2)$, where n is the total number of hops (links) along the preempting path, and m is the size of the set that contains all existing connections that have at least one link in common with the selected path for the preempting connection and having a priority less than that of the preempting connection.

In [8], we made a comparison study, in terms of overall network performance, between a decentralized algorithm that optimized the number of connections preempted, and the centralized algorithm by Garay and Gopal [2] which gives the best overall result among the algorithms they proposed. The study indicated that there is no significant performance difference between the two algorithms and, in fact, both performed very well.

1.2. Objectives

A centralized preemption scheme cannot fit well into decentralized/distributed networks (such as ATM PNNI (Private Network Node Interface) networks that have multiple routing domains [9]) because each control point has to make decisions and perform functions independent of other control points. For example, when a connection setup request is processed by a link manager and not enough resources are available, the link manager itself, independent of other link managers, has to select the connections to be preempted from the set of all connections currently using the link. This paper is the first known study of distributed connection preemption algorithms.

Based on the observations we made from a comprehensive simulation study of preemption in a general connection-oriented network setting, we developed two optimal decentralized/distributed connection preemption selection algorithms that minimize the disruption to existing connections while satisfying the constraints of higher priority connections. When preemption is necessary to establish a high priority connection, these algorithms are run locally by each link along the chosen path for the connection if bandwidth cannot be allocated on that link. Unlike the Garay and Gopal [2] algorithms which are heuristic, these two algorithms are locally optimal with respect to their respective objective functions over the parameters associated with an outgoing link: the bandwidth to be preempted, preemption of low-priority connections, and the number of connections to be preempted. These algorithms consider preemption at the link level and are run locally for each out-going link, that is, if a new end-to-end connection has to be established, each link along the chosen path of the new connection will cause a preemption algorithm to be executed at its control point residing on the node at the origin of the link if bandwidth cannot be allocated on that link. Thus, the algorithms are truly decentralized/distributed.

The first algorithm, named *Min_BW*, first minimizes the amount of bandwidth to be preempted at the link level, and if there is a choice of connections to be preempted with the above criterion, it chooses a combination of connections with the least priority, and if there is a choice of such combinations, it chooses a combination with the least number of connections. This objective function is most meaningful as a measure of the goodness of a preemption strategy because it optimizes first, the bandwidth preempted (which is a fine-grained and accurate measure of the amount of preemption), then the priority of connections preempted (which considers the relative importance of the amounts of bandwidth preempted in case of a tie), and lastly, the number of connections preempted (which is a crude measure of the amount of preemption). However, this algorithm has an exponential computational complexity of $O(k \cdot 2^k)$, where k is the number of connections sharing the link under consideration and having a priority less than that of the preempting connection. We would like to design an algorithm with polynomial complexity that approximates the behavior of *Min_BW*.

Based on our simulation study, from among the various algorithms that minimize the above three parameters (bandwidth preempted, number of connections preempted, priority of connections preempted) in different orders of preference, we identified and developed a polynomial algorithm that best approximates *Min_BW*. This second algorithm, named *Min_Conn*, first minimizes the number of connections to be preempted at the link level, then chooses the combination of connections to be preempted to minimize the bandwidth to be preempted, and if there is a choice of such combinations, it chooses a combination in which the connections have the least priority. This algorithm has a complexity of $O(k^2)$, where k is as defined for *Min_BW*.

Both *Min_BW* and *Min_Conn* are optimal with respect to their respective objective functions (although the objective function of *Min_BW* is more meaningful as a measure of goodness of a preemption strategy); their optimality is local because it uses the parameters on a per outgoing link basis. Henceforth, when we refer to optimality, we will implicitly mean local optimality.

We present a simulation study of the two algorithms and conclude that the polynomial algorithm *Min_Conn* performs almost as well as the exponential algorithm *Min_BW*. Our simulation study also provides useful insights into connection preemption and network dimensioning problems in order to achieve a desired level of network availability.

In summary, *Min_BW* is an algorithm that optimizes the criteria of (i) the bandwidth to be preempted, (ii) the priority of connections to be preempted, and (iii) the number of connections to be preempted, in that order. *Min_Conn* is an algorithm that optimizes the criteria of (i) the number of connections to be preempted, (ii) the bandwidth to be preempted, and (iii) the priority of connections to be preempted, in that order. Both are distributed/decentralized, i.e., consider preemptions at the link level. The algorithms are optimal with respect to their objective functions because they perform an exhaustive search of their search space to select a solution based on the criteria for which they claim optimality.

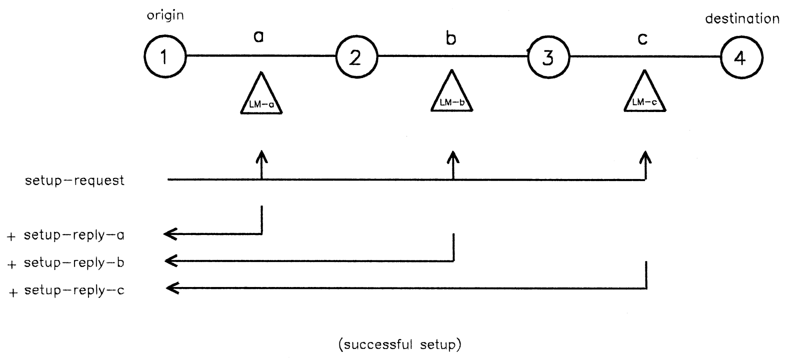
The rest of paper is organized as follows: Section 2 presents a simple connection control protocol for decentralized connection-oriented networks. Section 3 presents the two optimal algorithms along with an analysis of their complexity. Section 4 presents a simulation study of connection preemption and compares the performance of the two proposed connection preemption algorithms. Section 5 gives the conclusions.

2. Connection control protocol

We now present below a simple connection control protocol for decentralized connection-oriented networks. The main idea here is to introduce a general model which can be used for our discussions on the connection preemption problem. Our protocol borrows some concepts from NBBS (Networking Broadband Services) [4], a decentralized fast-packet network architecture, and ATM PNNI [9] which provide connection-oriented services using the concept of source routing and link state. That is, the source computes a complete route from the source to the destination based on its knowledge about the current states and utilizations of the links. Each link is owned by a link manager (LM), and when a significant link state change occurs, the link manager broadcasts the information to all the nodes in the network. There is no concept of centralized control, each link manager independent of other link managers decides whether it can accept a new connection when it receives a connection setup request.

A connection is setup as follows: The origin computes a complete route from the origin to the destination based on its knowledge about the current states and utilizations of the links. Then, the origin constructs a connection setup request for the connection and sends it to all the link managers along the computed route. A link manager along the route accepts the connection and returns a positive reply only if it can provide the resources to accommodate the connection. Otherwise, it rejects the connection and returns a negative reply to the origin. If a link manager accepts a connection, it allocates the requested resources for the connection. When the origin receives the replies it determines whether a connection setup is successful. The connection setup is successful only if all the replies are positive. If the connection setup is unsuccessful, the origin computes a new route (which excludes the links that replied unfavorably) and repeats the setup process. When the connection setup is unsuccessful, the origin also sends a path takedown request to the link managers along the path of the connection that replied favorably. When a link manager receives a path takedown request for a connection, it releases the network resources associated with that connection (Fig. 1).

When a setup request is processed by a link manager and not enough resources are available, the link manager selects connections to be preempted from the set of all connections that are currently using the link and whose priorities are lower than the priority of the requesting connection. Preemption is triggered at a link by the link manager only if enough resources can be released by preemption to accommodate the requesting connection at that link. For each connection to be preempted, the link manager sends a preemption notification message to the origin of the connection. At the receipt of preemption notification message, the origin takes some actions to reroute the connection. First, it takes down the connection by sending a path takedown request to all the link managers along the path of the connection (Fig. 2). Then, it computes a new route for the preempted connection and starts the setup process.



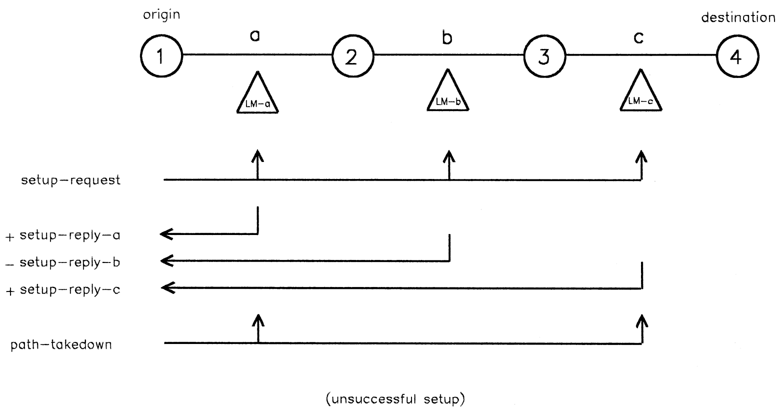


Fig. 1. Example of connection setup.

When a link or an intermediate node along the path of an ongoing connection fails, our protocol switches the connection to an alternate path. Link and node failures are detected by both origin and destination nodes via topology database update broadcasts. When a link or an intermediate node along the path of an ongoing connection fails, both the origin and destination send path takedown re-

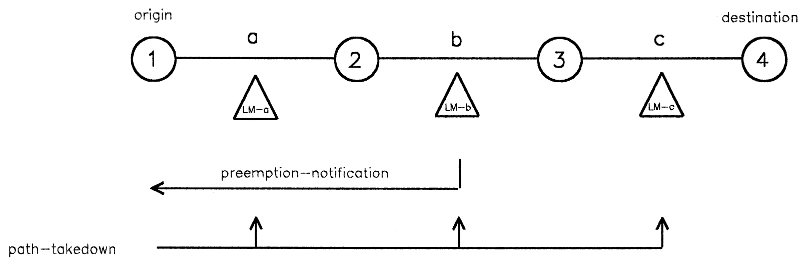


Fig. 2. Example of connection preemption.

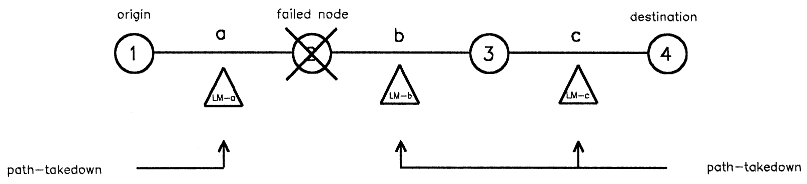


Fig. 3. Example of node failure.

quests along the path of the connection (Fig. 3). Then, the origin computes a new route (which excludes the failed links or nodes) and performs the connection setup process as described above.

When the origin or destination wants to terminate a connection, it constructs and sends a path takedown request to all the link managers along the path of the connection (Fig. 4).

2.1. Connection preemption: problem statement

Definition 1. A connection C_i is a two-tuple $C_i = (B_i, P_i)$, where
- B_i is the bandwidth requested by C_i.
- P_i is the priority of C_i. The priority of a connection is represented by a number greater than or equal to 1, with 1 being the lowest priority.

The connection preemption problem has to be solved in real-time because when a connection setup request arrives, frequently the holding time is not known and there is no knowledge about the future connection requests. Similarly, when a failure disrupts a connection, the connection must be rerouted immediately to provide reliable service. For the connection preemption problem, we therefore assume that a connection arrives with a predefined route, a predefined priority, and a predefined bandwidth. No knowledge of the future arrivals or the holding time is available.

Definition 2. Let
- $C_p = (B_p, P_p)$ be a new or rerouted connection,
- e_j be a link along the route of C_p without enough free bandwidth to accommodate C_p, and
- a_j be the free bandwidth in e_j.

C_p is a preempting connection if there exists a set of existing connections $C = \{C_1, C_2, \ldots, C_k\}$ that go through e_j such that if $C_i = (B_i, P_i) \in C$, where $1 \le i \le k$, then the following conditions hold: $P_p > P_i$ and $B_p \le a_j + \sum_{i=1}^{k} B_i$.

A connection has two parameters: bandwidth and priority. The set of connections to be preempted can be chosen by optimizing an objective function over these two parameters of the connections, and the number of connections to be preempted. In order to use a meaningful measure of goodness of a preemption strategy, the objective function of importance to the system has the following three parameters, in decreasing order of importance:
1. Preempt the least amount of bandwidth. The advantage of this strategy is that network bandwidth is better utilized and there is minimum disruption of user traffic. This is the primary objective.
2. Preempt the connections that have the least priority. This is a naive solution that can result in

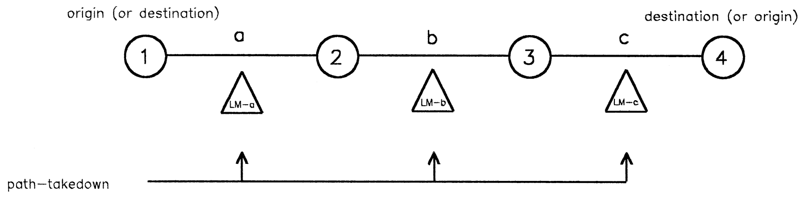


Fig. 4. Example of connection termination.

preemption of excessive bandwidth and connections. This is a secondary objective.

3. Preempt the least number of connections. The advantage of this strategy is that a minimum number of connections have to be preempted and rerouted. This is a tertiary objective.

Algorithm *Min_BW* achieves the above objective function. Unfortunately, *Min_BW* has an exponential complexity and we would like an algorithm with polynomial complexity that approximates the behavior of *Min_BW*. We examined all the other 5 permutations of the above parameters in search of such an algorithm, and based on an extensive simulation study of the resulting objective functions in a general network setting, we concluded that the objective function of algorithm *Min_Conn* best approximates the behavior of *Min_BW*. Therefore, *Min_BW* and *Min_Conn* are presented in Section 3.

3. Connection preemption algorithms

We present two connection preemption algorithms that optimize network performance by minimizing the disruption to the ongoing connections and by utilizing network bandwidth more efficiently. These algorithms are suited for both centralized and decentralized types of network. Our algorithms do not consider the complete route of the preempting connection nor do they examine the complete routes of the connections that share one or more links with the preempting connection. When connection preemption at a link is necessary to accommodate a new connection, the link manager at that link, independent of what other link managers along the path of the connection choose to do, selects a set of connections from the connections currently using the link for preemption. Thus, if preemption is necessary to setup the connection at two links (say a and b), the connections picked by link manager a are not necessarily the same as those picked by link manager b. When selecting connections for preemption, a link manager only considers the existing connections that go through its link and makes its selection only by examining connections' priorities and their requested bandwidth.

3.1. Algorithm Min_BW

Min_BW is an algorithm that optimizes the criteria of (i) the bandwidth to be preempted, (ii) the priority of connections to be preempted, and (iii) the number of connections to be preempted, in that order.

The algorithm returns the connections to be preempted in set P and works as follows: The algorithm evaluates the bandwidth to be preempted for each and every combination of connections, selected r at a time, starting with $r = 1$ and going upto $r = k$, where k is the number of connections using the link under consideration and having a priority less than that of the preempting connection. The **for** loop on line 5 varies the value of r. For any value of r, it is determined whether there is a combination of connections which when preempted will free up at least W bandwidth which is the bandwidth required tobe preempted. If there are two or more such combinations for a given value of r, then the combination with the lesser value of the sum of the bandwidths is selected. When there is more than one combination with the same amount of bandwidth, then the one with the lesser priorities is selected. This is achieved as follows for a given value of r. The loop on line 9 enumerates all combinations of r connections. Array A is a working array variable whose first r elements are indices, in ascending order, of connections being presently considered for preemption. Each iteration of the loop assigns to elements $A_1, A_2, \ldots, A_r$ a combination of values from 1 to k as follows: View the array as a car mileage odometer with the most significant position being A_1 and the least significant position being A_r, and which is incremented at each iteration according to the following rules:

- Elements of A are assigned values in the range 1 to k.
- The initial value of $A_i = i$, for $i = 1,2,\ldots,r$.
- The values of A are enumerated in ascending order.
- The value of $A_i > A_{i-1}$, for $i = 2,3,\ldots,r$.

The above rules imply that A_i, for $i = 1,2,\ldots,r$, can take values in the range i to $k - r + i$.

A **for** loop computes the aggregate bandwidth for each resulting enumeration of r combinations (line 11). The aggregate bandwidth for the combination that yields the least known bandwidth exceeding or equal to the bandwidth required to be preempted (W), is stored in *Min* at all times. The corresponding combination of connection indices is stored in array S (see lines 14,15).

When all combinations of connections for all values of r have been enumerated, line 26 places in set P, the optimal set of connections to be preempted. These are the connections whose indices are given by the elements in array S.

```
 1. $W := B_p - a_j$;
 2. Min$:= \sum_{i=1}^{k} B_i$;
 3. $S := C$;
 4. $P := \phi$;
 5. for $r = 1$ to $k$ do
 6.     for $l = 1$ to $r$ do
 7.         $A_l := l$;
 8.     endfor
 9.     for $m = 1$ to $k!/(k-r)!r!$ do
10.         Sum$:= 0$;
11.         for $l = 1$ to $r$ do
12.             Sum$:=$ Sum $+ B_{A_l}$;
13.         endfor
14.         if $W \le$ Sum $<$ Min
            then Min$:=$ Sum and $S \leftarrow A$;
15.         if $W \le$ Sum and Sum $=$ Min and
            $\sum_{m \in A} P_m < \sum_{n \in S} P_n$ then $S \leftarrow A$;
16.         $i := r$;
17.         while $i > 1$ and $A_i = k - r + i$ do
18.             $i := i - 1$;
19.         endwhile
20.         if $A_i < k - r + i$ then $A_i = A_i + 1$;
21.         for $l = i + 1$ to $r$ do
22.             $A_l := A_{l-1} + 1$;
23.         endfor
24.     endfor
25. endfor
26. $P := P \cup C_{S_i}$ for every index $i \in S$.
```

Example. For $k = 6$, $r = 3$, we show the values of A_1, A_2, A_3 separated by commas, as generated by the algorithm. Successive values of the three elements of A are separated by semicolons: 1,2,3; 1,2,4; 1,2,5; 1,2,6; 1,3,4; 1,3,5; 1,3,6; 1,4,5; 1,4,6; 1,5,6; 2,3,4; 2,3,5; 2,3,6; 2,4,5; 2,4,6; 2,5,6; 3,4,5; 3,4,6; 3,5,6; 4,5,6.

Complexity. For a given value of r, the main **for** loop of line 9 is executed $k!/(k-r)!r!$ times and each execution of the loop has $O(r)$ time complexity. We have

$$\sum_{r=1}^{k} r \frac{k!}{(k-r)!r!} = k \cdot 2^{k-1}.$$

Thus, the computational complexity is $O(k \cdot 2^k)$, where k is the number of connections using the link under consideration and having a priority less than that of the preempting connection. It can easily be shown that the problem of minimizing the amount of bandwidth to be preempted is NP-complete by showing a polynomial-time reduction to and from the knapsack problem. [3]

3.2. Algorithm Min_Conn

Min_Conn is an algorithm that optimizes the criteria of (i) the number of connections to be preempted, (ii) the bandwidth to be preempted, and (iii) the priority of connections to be preempted, in that order. The algorithm is distributed/decentralized, i.e., considers preemption at the link level and has a polynomial time computational complexity. The *Min_Conn* algorithm is shown below.

This algorithm returns the connections to be preempted in set P. Step 2 determines W, the amount of bandwidth that needs to be preempted in order to accommodate the preempting connection. Step 4 identifies the connection that has the smallest bandwidth which is greater than W, and if multiple such connections exist, it selects the connection with the least priority. If this step can identify a connection to be preempted, then only one connection has to be preempted. Otherwise, Step 10 performs a greedy method to ensure that a minimum number of connections are selected for preemption. Step 10 finds the largest bandwidth connection, and if there is more than one, selects the one with the lowest priority. It then removes the selected connection from C, the set of connections that are still using C, in Step 13, adds it to P in Step 14, and updates a_j, the amount of bandwidth available in the link e_j when the selected

[3] Knapsack problem: Given $C = \{C_1, C_2, \ldots, C_k\}$ and $W > 0$, $C_r > 0$, is there a subset of C such that the sum of the elements in the subset is W, i.e., $C_i + C_j + \ldots + C_l = W$? [6].

connection is preempted in Step 15. Steps 2 to 15 are executed in the **while** loop of line 1 until enough bandwidth to accommodate the preempting connection is freed.

```
1. while $B_p > a_j$ do
2.    $W := B_p - a_j$;
3.    $i := 0$;
4.    for $l = 1$ to $k$ do
5.       if $i = 0$ and $B_l \geq W$ then $i := l$;
6.       if $i > 0$ and $B_l \geq W$ and $(B_l < B_i$
         or $(B_l = B_i$ and $P_l < P_i))$ then $i := l$;
7.    endfor;
8.    if $i = 0$ then
9.       $i := 1$;
10.      for $l = 1$ to $k$ do
11.         if $B_l > B_i$ or $(B_l = B_i$ and $P_l < P_i)$
            then $i := l$;
12.      endfor;
13.   $C := C - \{C_i\}$;
14.   $P := P \cup \{C_i\}$;
15.   $a_j := a_j + B_i$.
16. endwhile
```

Upper bound on computational complexity: $O(k^2)$, where k is the number of connections using the link under consideration and having a priority less than that of the preempting connection. If the k connections are kept sorted, then the objective of *Min_Conn* can be achieved in $O(k \cdot \log k)$ computational complexity.

3.2.1. Optimality of the algorithms

Algorithms *Min_BW* and *Min_Conn* are both (locally) optimal with respect to their objective functions. This is because they perform an exhaustive search of their search space to select a solution based on the criteria for which they claim optimality.

The objective function of *Min_BW* is most meaningful to the system because it optimizes first, the bandwidth preempted (which is a fine-grained and accurate measure of the amount of preemption), then the priority of connections preempted (which considers the relative importance of the amounts of bandwidth preempted in case of a tie), and lastly, the number of connections preempted (which is a crude measure of the amount of preemption). However, *Min_BW* has an exponential complexity. Although *Min_Conn* optimizes a different objective function, we will show in Section 4 that its behaviour approximates that of *Min_BW* and it is a polynomial algorithm.

4. Simulation

4.1. Model

We used a connection-level simulation to study preemption and to compare the two proposed algorithms in a dynamic network environment where connections come and go. The simulation model has most mechanisms of typical connection-oriented networks. Its main components are a path selection algorithm which selects a minimum-hop path between an origin-destination pair, a connection setup and takedown protocol, and a topology information distribution protocol. In addition to the above components, the model also has a connection preemption protocol and a path-switch mechanism which reroutes connections preempted due to link/node failure or preemption. The simulation program is written in C and SIMSCRIPT and has about 5000 lines of code and consists of a number of processes which execute several dynamic objects and routines. A process is created at a simulated time and it performs a sequence of events separated by lapses of time. The process concept is used to represent connections, connection generation, and messages, while static objects such as route computation are represented using routines.

The input to the simulation program includes a network configuration — the nodes, the transmission links with their propagation delays and capacities,— source/destination distribution, connections' characteristics, link failure events, and other controlling parameters such as simulation time, simulation seeds, and maximum connection hops. The program collects and reports a number of statistics as will be described later.

The program simulates the lives of connections from the time they are created until they terminate. The flowchart in Fig. 5 shows a very high-level view of how a connection is handled in this simulation model.

Connection interarrival times are exponentially distributed. Upon arrival of a connection to the

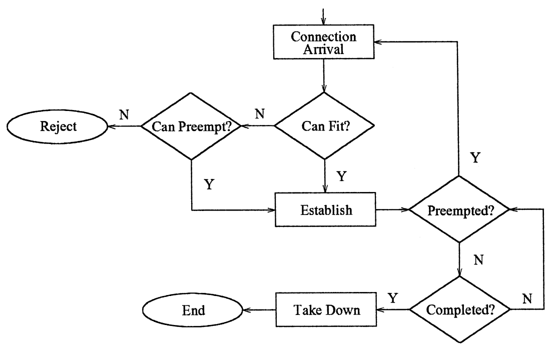


Fig. 5. Flowchart for simulation model.

network, its source and destination nodes, priority, bandwidth, holding time, and delay are chosen probabilistically. Once the connection's parameters are selected, a path selection algorithm is run and a path in the network is determined. This algorithm attempts to find a path that has a minimum number of hops while satisfying the connection's quality of service parameters. If there are several eligible paths with the same number of hops then one of them is chosen based on lowest ''weight'' of the path. This weight of a path is a measure of the load or the bandwidth allocated along the path. Specifically, each link in the network is associated with an increasing function of the load of the link; this function is termed the weight of the link. The weight of a path is the sum of weights of the individual links on the path. A path with a low weight indicates a path with a low load in terms of this measure. The path selection algorithm and the notion of link weights are described in detail in [11].

The connection control protocol described in Section 2 then attempts to establish the connection. Basically, when a connection request arrives, a connection is established if the network has the bandwidth to support the connection. Once established, the connection begins its ''talk'' phase. However, if there is not enough bandwidth to establish the connection, then if there are sufficient low-priority connections that can be preempted to free enough bandwidth for this connection, then those low-priority connections will be preempted and the connection request gets satisfied. When the connection request cannot be accommodated, it is rejected. When a connection is preempted, it is treated like a new connection. When a connection successfully completes its talk phase, it gets taken down. So, note that a successfully completed connection may have been rerouted one or more times due to preemption, link failure, or node failure.

Another feature included in the simulation model is that connections failed to set up (because there is no sufficient resource in at least one link along the path) can execute the path selection algorithm more than once. A parameter that limits the number of such attempts is defined as an input to the program. Each time the path selection algorithm is executed, links which responded negatively in the previous setup request are excluded from further consideration. If the setup request fails at every trial and the retry count exceeds the threshold, then the connection request is dropped.

Upon acceptance of a connection on a link or removal of a connection from a link, a bandwidth reservation table for that link is updated. When a significant change in the link bandwidth reservation occurs, a topology database update message is sent to every node in the network. This is done only if the change in the reservation level for the link is significant, i.e., if it exceeds some threshold value defined for that link. [4] A topology database update message is also broadcast when a link fails or comes up. So, a connection setup request may not be successful for two reasons: the topology database at the originating node may not be ''current'' and/or multiple sources may send connection setup requests to a particular link almost simultaneously, competing for a limited available bandwidth.

4.2. Experiments

Network structure: We have conducted wide-range simulation with various network conditions (i.e., network topology, number of priority levels, link bandwidth, traffic pattern, etc.) to study connection preemption and the behavior of the two proposed algorithms. In terms of network performance, most of our simulation experiments have indicated similar results. So, for the sake of brevity, in this paper, we present only one set of simulation experiments.

[4] This topology database update broadcast concept scheme is similar to the one specified in ATM PNNI [9].

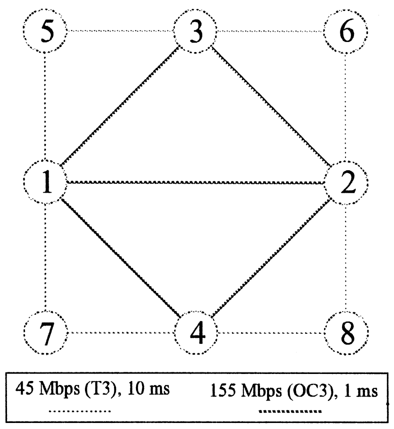


Fig. 6. Network topology and structure.

The following network model (which is an abstraction of a real network) was used in the simulation experiments. The network is two-tiered consisting of 8 nodes and 26 unidirectional links (Fig. 6). The inner links are OC3 links with a propagation delay of 1ms, and the outer links are T3 links with a propagation delay of 10 ms. In the experiments, the origin and destination pairs for the connections were selected such that the load in the network is uniformly distributed (as shown in Table 1).

Traffic profile: Table 2 shows the traffic profile used in the simulation experiments. Many connection types in terms of bandwidth size, holding time, and delay requirement were used along with three priority levels: low, medium, and high. The distribution of the priority levels is uniform, i.e., on the average the number of connection requests with low priority is the same as the number of connection requests with medium or high priority. The bandwidth range for connections is between 64 Kbps to 4000 Kbps. The distribution of bandwidth within this range is also uniform. The connections' holding times are assumed to be exponentially distributed with a mean of 600 seconds. The propagation delay is the end-to-end delay for data travel imposed by links along the path and reflects the application's constraints. The propagation delay range for connections is between 10 ms to 60 ms with a uniform distribution. Propagation delay is an important parameter, particularly for real-time applications such as voice and video. Note that the delay requirement of some connections will have impact on route selection, that is, given that bandwidth is available, not every route can meet the delay requirements of certain connections.

Table 1
Network load distribution

Nodes	Origin probability	Destination probability (given that origin $\neq$ destination)
1,2	0.25	0.25
3,4	0.125	0.125
5,6,7,8	0.0625	0.0625

Performance metrics: Our simulation program collects statistics about a number of performance measures that indicate how well the network performs with a particular connection preemption algorithm. These measures are averaged over the life of simulation. The following three metrics are considered in this study:

1. Connection success probability: This is the probability that a connection of a given priority is successfully established and completes its talk time (holding time). Note that a successfully completed connection may have been rerouted one or more times due to preemption or link/node failures.
2. Connection preempting probability: This is the probability that a connection of a given priority preempts one or more connections.
3. Connection reroute probability: This is the probability that a connection of a given priority is rerouted one or more times due to preemption or link/node failures.

Nature of experiments: The structure of the simulation experiments is as follows. Each experiment consists of 24 independent runs, and 95% confidence intervals are obtained for all performance measures. The reason for doing this many runs was to be able to get good confidence intervals. The independence of the runs was achieved by shuffling the seeds required by the program. Since the runs are independent, we can assume identical distribution of the replications. Thus, the central limit theorem can be used to justify the use of Gaussian statistics to construct confidence intervals on the performance measures. As the number of replications is small, we can assume that the mean is distributed as student-t distribution and calculate the $100(1-\alpha)\%$ confi-

Table 2
Traffic profile

Connection priority	Bandwidth range (in Kbps) (uniform distribution)	Mean holding time (in seconds) (exponential distribution)	Propagation delay (in ms) (uniform distribution)
Low (1)	64–4000	600	10–60
Medium (2)	64–4000	600	10–60
High (3)	64–4000	600	10–60

dence interval on the mean from the replications using [1]:

$$\bar{x} - t_{\alpha/2}\frac{s}{\sqrt{r}} < \mu < \bar{x} + t_{\alpha/2}\frac{s}{\sqrt{r}},$$

where $\bar{x}$ is the sample mean, μ is the true mean, s is the sample standard deviation, r is the number of replications, and $t_{\alpha/2}$ is the critical value of the student-t distribution with $(r-1)$ degrees of freedom. [5] The confidence intervals were observed to be too small to report here for the simplicity of presentation. Each simulation run is 10,000 seconds of simulation time. To observe the impacts of nonstationary network conditions, link failure is introduced. Specifically, the pair of bidirectional links that connect nodes 1 and 2 were failed. Note that this link failure causes a lot of connections to be rerouted and places the network under heavy stress because these links are the high-speed backbone links.

Network behavior as a function of time: Fig. 7 shows the connection success probability and the average link reservation level versus the simulation time for one of the experiments. The connection arrival rate to the network for this experiment has an exponential distribution with a mean of about 1.5 connections/second. Initially, there is no connection in the network, so the average link reservation level is zero. As connections arrive, the average link reservation level goes up and the connection success probability drops. After a short time of about 1500 seconds, the average link reservation level and the connection success probability reach a stable level, about 0.98 for the connection success probability and about 81% for the average link reservation level. Once the network reached a stable level, the pair of links that connect nodes 1 and 2 are made to fail. When this happens, the connection success probability drops and the average link reservation level goes up. This is expected as the network has less bandwidth for new and ongoing connections. With link failure, the network again reaches a stable level but the connection success probability is now much lower at 0.82, and the average link reservation level is about 0.90. After 5000 seconds, when the links are brought up again, the network goes back to its normal operating points.

Connection success probability as a function of arrival rate: Fig. 8 shows the connection success probability as a function of the connection arrival rate with and without preemption. This figure allows the comparison of the two preemption algorithms and the analysis of the effect of preemption in terms of network throughput. The top curve is for the *Min_BW* algorithm, the middle curve is for the *Min_Conn* algorithm, and the bottom curve is for the non-preemption case. In all the cases, we see that

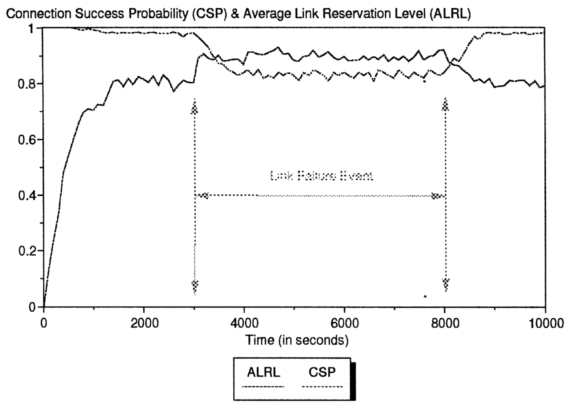


Fig. 7. Network behavior.

[5] For this set of experiments we obtained the 95% confidence intervals for all performance measures. The $t_{\alpha/2}$ value for 95% confidence interval when $r = 24$ is 2.069 which is obtained from [3].

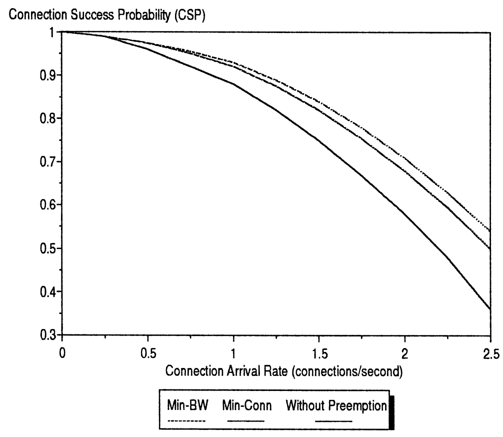


Fig. 8. Preemption versus no preemption.

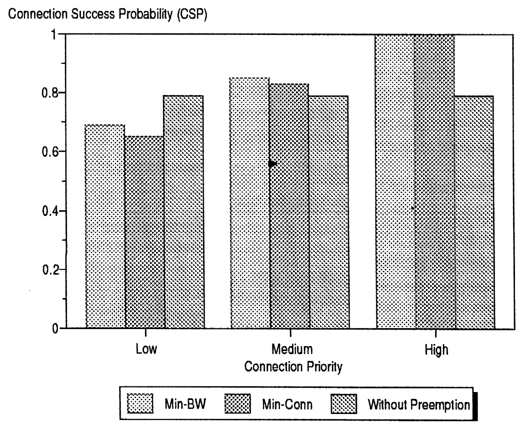


Fig. 9. Connection success probability.

when the connection arrival rate is low, i.e., when the network is lightly loaded, the connection success probability is almost one. As the connection arrival rate increases, the connection success probability decreases because the network now has less free bandwidth. When the connection arrival rate is low, the effect of preemption is very small, i.e., there is no significant difference between the preemption and the non-preemption cases in terms of the connection success probability. However, as the load in the network increases due to more connection requests, the advantage of preemption becomes more obvious, as more connections get through. This is because when preemption is coupled with path-switch, it improves the path selection process by providing a way to correct ''wrong'' decisions made in the past due to lack of knowledge about the future connection request arrival pattern. As far as the performance of these two connection preemption selection algorithms is concerned, there is no significant difference. Basically, the *Min_BW* algorithm performs a little better than the *Min_Conn* algorithm and the performance difference increases with the connection arrival rate. The reason for this performance difference is that the *Min_BW* algorithm causes less preemption of excess bandwidth than the *Min_Conn* algorithm, allowing more efficient utilization of network bandwidth.

Performance metrics for priority levels: We now look at performance measures by considering how individual priorities are affected with and without preemption.

Fig. 9 shows the connection success probability for each priority level with and without preemption for one of the experiments. The figure is for the case when the connection arrival rate is about 1.5 connections/second. This results in a connection success probability of about 0.8 without preemption, so the network is under stress. When preemption is used, we see that for both the algorithms the connection success probability increases nicely with the connection's priority. The higher network availability for connections of greater priority is the expected result from the use of preemption. Although the *Min_BW* does a little better than *Min_Conn* for the connection success probability achieved for different priority levels, there is no significant difference. Without

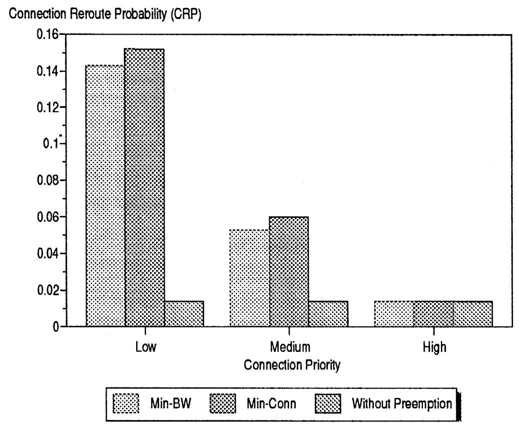


Fig. 10. Connection reroute probability.

preemption, the connection success probability is the same for all priority levels.

Fig. 10 shows the connection reroute probability for each priority level with and without preemption, for one of the experiments. The connection arrival rate is the same as before, i.e., about 1.5 connections/second, indicating that the network is under stress. With preemption, we see that for both the algorithms the connection reroute probability decreases as the connection's priority increases. This is very desirable as we want the high priority connections to be disturbed less. Without preemption, connections are rerouted only due to link failure. Hence, the connection reroute probability is lower without preemption. It is observed that with or without preemption, the connection reroute probability for the high-priority connections is the same. This is expected because the high-priority connections do not get preempted due to preemption. Without preemption, of course, the connection reroute probability is the same for all priority levels. A nice observation is that even with preemption, the overall connection reroute probability is low. In terms of the connection reroute probability, there is no significant difference between the two connection preemption selection algorithms *Min_Conn* and *Min_BW*, although *Min_BW* performs a little better.

Fig. 11 shows the connection preempting probability for each priority level for one of the experiments. The connection preempting probability is zero without preemption and it is also zero for the low priority connections. The connection arrival rate is the same as before, i.e., about 1.5 connections/second. It is interesting to observe that the connection preempting probability is higher for the medium priority connections than for the high priority connections – this is due to the cascading effect of preemption. This cascading effect of preemption can be explained as follows. When a connection of priority i preempts a connection of priority $i-1$, if the connection of priority $i-1$ is not the lowest priority connection, it tries to get reestablished and in this process it might preempt connections of priority $i-2$. Therefore, connections which are not of the highest priority can actually have higher preempting probability than the connections of the highest priority. But overall it is observed that the connection preempting probability is low even when the network is heavily loaded. Both the algorithms *Min_BW* and *Min_Conn* perform very well in terms of the connection preempting probability and there is no significant performance difference between them.

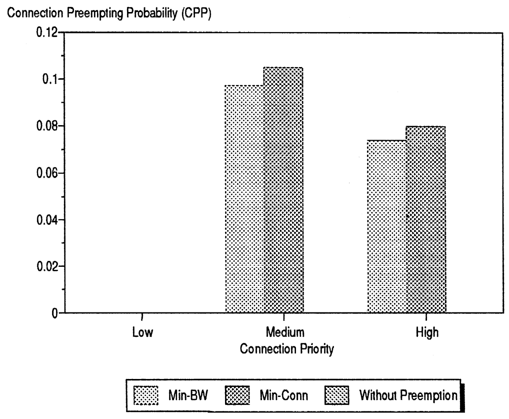


Fig. 11. Connection preempting probability.

5. Conclusions

In this paper, we investigated the connection preemption problem and presented a comprehensive simulation study of preemption in a general decentralized/distributed connection-oriented network setting. [6] We observed that connection preemption when coupled with the capability to reroute connections (preempted due to failure or preemption) provides higher network availability to high-priority connections and utilizes network bandwidth more efficiently, allowing more connections to get through. This is especially useful during nonstationary network conditions when demand for network bandwidth is higher. Our simulation study also provided insights into connection preemption and network dimensioning problems in order to achieve a desired level of network availability.

We first proposed and studied algorithm *Min_BW* which optimizes the criteria of (i) the bandwidth to be preempted, (ii) the priority of connections to be preempted, and (iii) the number of connections to be preempted, in that order. This objective function is

[6] To the best of our knowledge, this is the only study of connection preemption in a decentralized/distributed network setting.

most meaningful to the system as a measure of the goodness of a preemption strategy because it optimizes first, the bandwidth preempted (which is a fine-grained and accurate measure of the amount of preemption), then the priority of connections preempted (which considers the relative importance of the amounts of bandwidth preempted in case of a tie), and lastly, the number of connections preempted (which is a crude measure of the amount of preemption). However, *Min_BW* has an exponential complexity. Based on the observations made in an extensive simulation study with various connection preemption schemes, we then designed algorithm *Min_Conn* that optimizes a different objective function, best approximates the behavior of *Min_BW* among the 5 other objective functions defined by permutations of the above three parameters, and has a polynomial computational complexity. Algorithm *Min_Conn* optimizes the criteria of (i) the number of connections to be preempted, (ii) the bandwidth to be preempted, and (iii) the priority of connections to be preempted, in that order. Both algorithms are decentralized/distributed, i.e., consider preemption at the link level. From a comparison study of these two algorithms, we concluded that, in terms of overall network performance, there is no significant performance difference between the two, however, in terms of computational complexity, *Min_Conn* is polynomial while *Min_BW* is exponential. In terms of overall network performance, the *Min_BW* algorithm performs a little better than the *Min_Conn* algorithm and the performance difference increases as the load in the network increases. The main reason for this performance difference is that *Min_BW* minimizes preemption of excess bandwidth, allowing more efficient utilization of network bandwidth. Given that connection preemption is a real-time problem, the polynomial time algorithm (i.e., *Min_Conn*) is more favorable.

References

[1] W.P. Lovegrove, J.L. Hammond, Simulation methods for studying nonstationary behavior of computer networks, IEEE J. Select. Areas Comm. 8 (9) (1990) 1696–1708.

[2] J.A. Garay, I.S. Gopal, Connection preemption in communication networks, in: Proc. Infocom '92, 1992, pp. 1043–1050.

[3] R.E. Walpole, R.H. Myers, Probability and Statistics for Engineers and Scientists, Macmillan, New York, 1985.

[4] Networking Broadband Services Architecture (special issue), IBM Syst. J. 34 (4) (1995).

[5] D. Sleator, R. Tarjan, Amortized efficiency of list update and paging rules, Comm. ACM 28 (2) (1985) 202–208.

[6] M.R. Garey, D.S. Johnson, Computers and Interactability, W.H. Freeman, San Francisco, 1979.

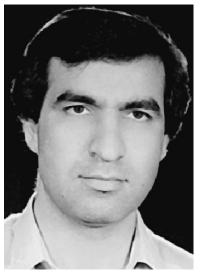
[7] M. Wernik, O. Aboul Magd, H. Gilbert, Traffic management for B-ISDN services, IEEE Network 6 (5) (1992) 10–19.

[8] M. Peyravian, Providing different levels of network availability in high-speed networks, in: Proc. Globecom '94, 1994, pp. 941–945.

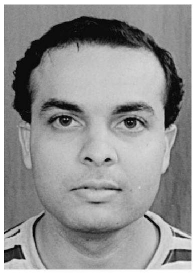
[9] PNNI Draft Specification, ATM Forum 95-0471R14, December 1995.

[10] D.E. McDysan, D.L. Spohn, ATM: Theory and Application, McGraw-Hill, New York, 1994.

[11] L. Gün, R. Guérin, Bandwidth management and congestion control framework of the broadband network architecture, Comput. Netw. ISDN Syst. 26 (1) (1993) 61–78.



Mohammad Peyravian is a senior scientist at IBM, Research Triangle Park, where he is involved with protocol and algorithm design for high-speed networking technologies. He received the B.S. and M.S. degrees from the University of Miami and the Ph.D. degree from the Georgia Institute of Technology in Electrical Engineering in 1987, 1989, and 1992, respectively. He does research in the areas of networking, telecommunications, and cryptography. Dr. Peyravian has published papers extensively in various journals and proceedings. He has received several IBM invention and technical achievement awards. Dr. Peyravian is a member of the IEEE computer and communications societies, Tau Beta Pi, and Eta Kappa Nu.



Ajay Kshemkalyani is an Assistant Professor in Electrical and Computer Engineering and Computer Science at the University of Cincinnati since September 1997. From 1991 to 1997 he worked at IBM Research Triangle Park in distributed systems and computer networks. He received a Ph.D. in Computer and Information Science from the Ohio State University in 1991, and a B.Tech. in Computer Science and Engineering from the Indian Institute of Technology, Bombay, in 1987. His current research interests are in distributed computing, networking, and operating systems. He is a member of the Association for Computing Machinery and a senior member of the IEEE.